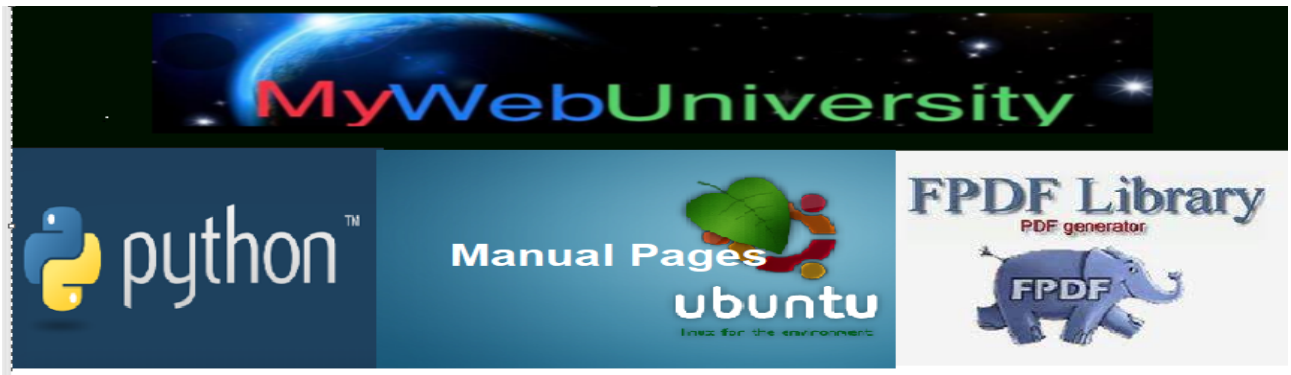

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'perf-dlfilter.1'

$ man perf-dlfilter.1

PERF-DLFILTER(1) perf Manual PERF-DLFILTER(1)

NAME

perf-dlfilter - Filter sample events using a dynamically loaded shared object file

SYNOPSIS

perf script [--dlfilter file.so] [--dlarg arg]...

DESCRIPTION

This option is used to process data through a custom filter provided by a dynamically loaded shared object file. Arguments can be passed using --dlarg and retrieved using perf_dlfilter_fns.args().

If file.so does not contain "/", then it will be found either in the current directory, or perf tools exec path which is ~/libexec/perf-core/dlfilters for a local build and install (refer perf --exec-path), or the dynamic linker paths.

API

The API for filtering consists of the following:

```
#include <perf/perf_dlfilter.h>
struct perf_dlfilter_fns perf_dlfilter_fns;
int start(void **data, void *ctx);
int stop(void *data, void *ctx);
int filter_event(void *data, const struct perf_dlfilter_sample *sample, void *ctx);
int filter_event_early(void *data, const struct perf_dlfilter_sample *sample, void *ctx);
const char *filter_description(const char **long_description);
```

If implemented, start will be called at the beginning, before any calls to filter_event or filter_event_early. Return 0 to indicate success, or return a negative error code. *data can

be assigned for use by other functions. ctx is needed for calls to perf_dlfilter_fns, but most perf_dlfilter_fns are not valid when called from start.

If implemented, stop will be called at the end, after any calls to filter_event or filter_event_early. Return 0 to indicate success, or return a negative error code. data is set by start. ctx is needed for calls to perf_dlfilter_fns, but most perf_dlfilter_fns are not valid when called from stop.

If implemented, filter_event will be called for each sample event. Return 0 to keep the sample event, 1 to filter it out, or return a negative error code. data is set by start. ctx is needed for calls to perf_dlfilter_fns.

filter_event_early is the same as filter_event except it is called before internal filtering.

If implemented, filter_description should return a one-line description of the filter, and optionally a longer description.

Do not assume the sample argument is valid (dereferenceable) after filter_event and filter_event_early return.

Do not assume data referenced by pointers in struct perf_dlfilter_sample is valid (dereferenceable) after filter_event and filter_event_early return.

The perf_dlfilter_sample structure

filter_event and filter_event_early are passed a perf_dlfilter_sample structure, which contains the following fields:

```
/*
 * perf sample event information (as per perf script and <linux/perf_event.h>)
 */
struct perf_dlfilter_sample {
        __u32 size; /* Size of this structure (for compatibility checking) */
        __u16 ins_lat;          /* Refer PERF_SAMPLE_WEIGHT_TYPE in <linux/perf_event.h> */
        __u16 p_stage_cyc;      /* Refer PERF_SAMPLE_WEIGHT_TYPE in <linux/perf_event.h> */
        __u64 ip;
        __s32 pid;
        __s32 tid;
        __u64 time;
        __u64 addr;
        __u64 id;
```

```
        __u64 stream_id;
        __u64 period;
        __u64 weight;          /* Refer PERF_SAMPLE_WEIGHT_TYPE in <linux/perf_event.h> */
        __u64 transaction;     /* Refer PERF_SAMPLE_TRANSACTION in <linux/perf_event.h> */
        __u64 insn_cnt; /* For instructions-per-cycle (IPC) */
        __u64 cyc_cnt;         /* For instructions-per-cycle (IPC) */
        __s32 cpu;
        __u32 flags;           /* Refer PERF_DLFILTER_FLAG_* above */
        __u64 data_src;        /* Refer PERF_SAMPLE_DATA_SRC in <linux/perf_event.h> */
        __u64 phys_addr;       /* Refer PERF_SAMPLE_PHYS_ADDR in <linux/perf_event.h> */
        __u64 data_page_size;  /* Refer PERF_SAMPLE_DATA_PAGE_SIZE in <linux/perf_event.h> */
        __u64 code_page_size;  /* Refer PERF_SAMPLE_CODE_PAGE_SIZE in <linux/perf_event.h> */
        __u64 cgroup;          /* Refer PERF_SAMPLE_CGROUP in <linux/perf_event.h> */
        __u8  cpumode;         /* Refer CPUMODE_MASK etc in <linux/perf_event.h> */
        __u8  addr_correlates_sym; /* True => resolve_addr() can be called */
        __u16 misc;            /* Refer perf_event_header in <linux/perf_event.h> */
        __u32 raw_size;        /* Refer PERF_SAMPLE_RAW in <linux/perf_event.h> */
        const void *raw_data;  /* Refer PERF_SAMPLE_RAW in <linux/perf_event.h> */
        __u64 brstack_nr;      /* Number of brstack entries */
        const struct perf_branch_entry *brstack; /* Refer <linux/perf_event.h> */
        __u64 raw_callchain_nr; /* Number of raw_callchain entries */
        const __u64 *raw_callchain; /* Refer <linux/perf_event.h> */
        const char *event;
        __s32 machine_pid;
        __s32 vcpu;
    };
```

Note: machine_pid and vcpu are not original members, but were added together later. size can be used to determine their presence at run time. PERF_DLFILTER_HAS_MACHINE_PID will be defined if they are present at compile time. For example:

```
    #include <perf/perf_dlfilter.h>
    #include <stddef.h>
    #include <stdbool.h>
    static inline bool have_machine_pid(const struct perf_dlfilter_sample *sample)
```

```
{
#ifdef PERF_DLFILTER_HAS_MACHINE_PID
        return sample->size >= offsetof(struct perf_dlfilter_sample, vcpu) + sizeof(sample->vcpu);
#else
        return false;
#endif
}
```

The perf_dlfilter_fns structure

The perf_dlfilter_fns structure is populated with function pointers when the file is loaded.

The functions can be called by filter_event or filter_event_early.

```
struct perf_dlfilter_fns {
        const struct perf_dlfilter_al *(*resolve_ip)(void *ctx);
        const struct perf_dlfilter_al *(*resolve_addr)(void *ctx);
        char **(*args)(void *ctx, int *dlargc);
        __s32 (*resolve_address)(void *ctx, __u64 address, struct perf_dlfilter_al *al);
        const __u8 *(*insn)(void *ctx, __u32 *length);
        const char *(*srcline)(void *ctx, __u32 *line_number);
        struct perf_event_attr *(*attr)(void *ctx);
        __s32 (*object_code)(void *ctx, __u64 ip, void *buf, __u32 len);
        void (*al_cleanup)(void *ctx, struct perf_dlfilter_al *al);
        void *(*reserved[119])(void *);
};
```

resolve_ip returns information about ip.

resolve_addr returns information about addr (if addr_correlates_sym).

args returns arguments from --dlarg options.

resolve_address provides information about address. al?size must be set before calling.

Returns 0 on success, -1 otherwise. Call al_cleanup() (if present, see below) when al data

is no longer needed.

insn returns instruction bytes and length.

srcline return source file name and line number.

attr returns perf_event_attr, refer <linux/perf_event.h>.

object_code reads object code and returns the number of bytes read.

al_cleanup must be called (if present, so check perf_dlfilter_fns.al_cleanup != NULL) after

resolve_address() to free any associated resources.

Do not assume pointers obtained via perf_dlfilter_fns are valid (dereferenceable) after filter_event and filter_event_early return.

The perf_dlfilter_al structure

The perf_dlfilter_al structure contains information about an address.

```
/*
 * Address location (as per perf script)
 */
struct perf_dlfilter_al {
        __u32 size; /* Size of this structure (for compatibility checking) */
        __u32 symoff;
        const char *sym;
        __u64 addr; /* Mapped address (from dso) */
        __u64 sym_start;
        __u64 sym_end;
        const char *dso;
        __u8  sym_binding; /* STB_LOCAL, STB_GLOBAL or STB_WEAK, refer <elf.h> */
        __u8  is_64_bit; /* Only valid if dso is not NULL */
        __u8  is_kernel_ip; /* True if in kernel space */
        __u32 buildid_size;
        __u8 *buildid;
        /* Below members are only populated by resolve_ip() */
        __u8 filtered; /* true if this sample event will be filtered out */
        const char *comm;
        void *priv; /* Private data. Do not change */
};
```

Do not assume data referenced by pointers in struct perf_dlfilter_al is valid (dereferenceable) after filter_event and filter_event_early return.

perf_dlfilter_sample flags

The flags member of perf_dlfilter_sample corresponds with the flags field of perf script.

The bits of the flags are as follows:

```
/* Definitions for perf_dlfilter_sample flags */
enum {
```

```
        PERF_DLFILTER_FLAG_BRANCH       = 1ULL << 0,
        PERF_DLFILTER_FLAG_CALL         = 1ULL << 1,
        PERF_DLFILTER_FLAG_RETURN       = 1ULL << 2,
        PERF_DLFILTER_FLAG_CONDITIONAL  = 1ULL << 3,
        PERF_DLFILTER_FLAG_SYSCALLRET   = 1ULL << 4,
        PERF_DLFILTER_FLAG_ASYNC        = 1ULL << 5,
        PERF_DLFILTER_FLAG_INTERRUPT    = 1ULL << 6,
        PERF_DLFILTER_FLAG_TX_ABORT     = 1ULL << 7,
        PERF_DLFILTER_FLAG_TRACE_BEGIN  = 1ULL << 8,
        PERF_DLFILTER_FLAG_TRACE_END    = 1ULL << 9,
        PERF_DLFILTER_FLAG_IN_TX        = 1ULL << 10,
        PERF_DLFILTER_FLAG_VMENTRY      = 1ULL << 11,
        PERF_DLFILTER_FLAG_VMEXIT       = 1ULL << 12,
    };
```

EXAMPLE

Filter out everything except branches from "foo" to "bar":

```
    #include <perf/perf_dlfilter.h>
    #include <string.h>
    struct perf_dlfilter_fns perf_dlfilter_fns;
    int filter_event(void *data, const struct perf_dlfilter_sample *sample, void *ctx)
    {
        const struct perf_dlfilter_al *al;
        const struct perf_dlfilter_al *addr_al;
        if (!sample->ip || !sample->addr_correlates_sym)
            return 1;
        al = perf_dlfilter_fns.resolve_ip(ctx);
        if (!al || !al->sym || strcmp(al->sym, "foo"))
            return 1;
        addr_al = perf_dlfilter_fns.resolve_addr(ctx);
        if (!addr_al || !addr_al->sym || strcmp(addr_al->sym, "bar"))
            return 1;
        return 0;
    }
```

To build the shared object, assuming perf has been installed for the local user i.e. perf_dlfilter.h is in ~/include/perf :

```
gcc -c -I ~/include -fpic dlfilter-example.c
gcc -shared -o dlfilter-example.so dlfilter-example.o
```

To use the filter with perf script:

```
perf script --dlfilter dlfilter-example.so
```

NOTES

The dlfilter .so file will be dependent on shared libraries. If those change, it may be necessary to rebuild the .so. Also there may be unexpected results if the .so uses different versions of the shared libraries that perf uses. Versions can be checked using the ldd command.

SEE ALSO

perf-script(1)