textsearch infrastructure + skb_find_text()

lwn.net/Articles/135159/

From:	Thomas Graf <tgraf@suug.ch>
To:	netdev@oss.sgi.com
Subject:	[RFC] textsearch infrastructure + skb_find_text()
Date:	Thu, 5 May 2005 01:40:36 +0200
Cc:	Pablo Neira <pablo@eurodev.net>
Archive-link:	Article, Thread

```
The patch below is a report on the current state of the textsearch
infrastructure and its first user skb_find_text(). The textsearch
is kept as simple as possible but advanced enough to handle non-linear
data such as skb fragments. Unlike in many other approaches the text
input is not seen as a single pointer but rather as a continuously
called callback get_text() until 0 is returned allowing to search
on any kind of data and to implement customized from-to limits.

The patch is separated into 3 parts, the first one being the textsearch
infrastructure itself followed by a simple Knuth-Morris-Pratt
implementation for reference. I'm also working on what could be called
the smallest regular expression implementation ever but I left that
out for now since it still has issues. Last but not least the
function skb_find_text() written in a hurry and probably not yet
correct but you should get the idea. From a userspace perspective
the first user will be an ematch but writing it will be peanuts
so I left it out for now.

Basically what it looks like right now is:

int pos;
struct ts_state;
struct ts_config *conf = textsearch_prepare("kmp", "hanky", 5, GFP_KERNEL, 1);

/* search for "hanky" at offset 20 until end of packet */
for (pos = skb_find_text(skb, 20, INT_MAX, conf, &state;
     pos >= 0;
     pos = textsearch_next(conf, &state)) {
        printk("Need a hanky? I found one at offset %d.\n", pos);
}

textsearch_put(conf);
kfree(conf);

You might wonder about the 1 given to _prepare(),  it indicates whether
to autoload modules because the ematches will need it to be able to drop
rtnl sem.

The code is not tested and cerainly not bug free yet but should compile.
```

Thoughts?

```
diff -X dontdiff -Nru linux-2.6.12-rc3.orig/include/linux/textsearch.h
linux-2.6.12-rc3/include/linux/textsearch.h
--- linux-2.6.12-rc3.orig/include/linux/textsearch.h 1970-01-01 01:00:00.000000000
+0100
+++ linux-2.6.12-rc3/include/linux/textsearch.h 2005-05-05 00:35:07.000000000
+0200
@@ -0,0 +1,169 @@
+#ifndef __LINUX_TEXTSEARCH_H
+#define __LINUX_TEXTSEARCH_H
+
+#ifdef __KERNEL__
+
+#include <linux/types.h>
+#include <linux/list.h>
+#include <linux/kernel.h>
+#include <linux/module.h>
+#include <linux/err.h>
+
+struct ts_config;
+
+/**
+ * struct ts_state - textsearch state
+ * @offset: current offset for next match
+ * @args: for persistant variables of get_text()
+ */
+struct ts_state
+{
+ int   offset;
+ long   args[6];
+};
+
+/**
+ * struct ts_ops - textsearch operations
+ * @name: name of search algorithm
+ * @init: called upon initialization to prepare a search
+ * @find: does the actual matching on a prepared configuration
+ * @owner: reference to algorithm module if existent
+ * @list: operations list we are on
+ */
+struct ts_ops
+{
+ const char  *name;
+ struct ts_config * (*init)(const unsigned char *, size_t, int);
+ int   (*find)(struct ts_config *,
+     struct ts_state *);
+ struct module  *owner;
+ struct list_head list;
+};
+
+/**
+ * struct ts_config - textsearch configuration
+ * @ops: textsearch operations
+ * @get_text: callback to fetch text to search in
+ * @len: length of pattern
+ */
+struct ts_config
+{
+ struct ts_ops  *ops;
+
+ /**
+  * get_text - return next chunk of text
+  * @offset: Number of bytes consumed by the matcher
+  * @dst: destination buffer
+  * @conf: search configuration
+  * @state: search state
+  *
+  * Gets called repeatedly until 0 is returned.Must assign a pointer
+  * to the start of the next chunk of text to *dst and return the
+  * length of the chunk or 0 if at the end. offset==0 indicates
+  * a new search. May store/read persistant values in state->args[].
+  */
+ int   (*get_text)(int offset, unsigned char **dst,
+         struct ts_config *conf,
+         struct ts_state *state);
+
+ /**
+  * finish - called when the matching has been completed successful
+  *          or not.
+  * @conf: search configuration
+  * @state: search state
+  */
+ void   (*finish)(struct ts_config *conf,
+       struct ts_state *state);
+ int   pattern_len;
+};
+
+/* Do no use this function directly */
+static inline int __textsearch_find(struct ts_config *conf,
+        struct ts_state *state)
+{
+ int ret = conf->ops->find(conf, state);
+
+ if (conf->finish)
+  conf->finish(conf, state);
+
+ return ret;
+}
+
+/**
+ * textsearch_find - search for a pattern in a text
+ * @conf: search configuration
+ * @state: search state
+ *
+ * Peforms the actual search on the prepared configuration.
+ *
+ * Returns the position of first occurrence of the pattern or a
+ *         negative number if no match was found.
+ */
+static inline int textsearch_find(struct ts_config *conf,
+       struct ts_state *state)
```

```
+{
+ state->offset = 0;
+ return __textsearch_find(conf, state);
+}
+
+/**
+ * textsearch_next - continue search for a pattern in a text
+ * @conf: search configuration
+ * @state: search state
+ *
+ * Continues a search looking for more occurrences of the pattern.
+ * You must call textsearch_find() to search the first occurrence
+ * in order to reset the state.
+ *
+ * Returns the position of next occurance of the pattern or a
+ *         negative number if no match was found.
+ */
+static inline int textsearch_next(struct ts_config *conf,
+      struct ts_state *state)
+{
+ state->offset += conf->pattern_len;
+ return __textsearch_find(conf, state);
+}
+
+/**
+ * textsearch_put - give back a textsearch configuration
+ * @conf: search configuration
+ *
+ * Releases all references of the configuration. Must be
+ * called prior to freeing the object.
+ */
+static inline void textsearch_put(struct ts_config *conf)
+{
+ if (conf->ops)
+  module_put(conf->ops->owner);
+}
+
+/**
+ * alloc_ts_config - allocate a textsearch configuration
+ * @payload: size of additional module specific data required
+ * @gfp_mask: allocation mask
+ *
+ * Returns a new, empty textsearch configuration or a ERR_PTR().
+ */
+static inline struct ts_config *alloc_ts_config(size_t payload, int gfp_mask)
+{
+ struct ts_config *conf;
+
+ conf = kmalloc(sizeof(*conf) + payload, gfp_mask);
+ if (conf == NULL)
+  return ERR_PTR(-ENOMEM);
+
+ memset(conf, 0, sizeof(*conf) + payload);
+ return conf;
+}
+
+extern int textsearch_register(struct ts_ops *);
```

```
+extern int textsearch_unregister(struct ts_ops *);
+extern struct ts_config *textsearch_prepare(const char *, const unsigned char
*,
+         size_t, int, int);
+extern int textsearch_find_continuous(struct ts_config *, struct ts_state *,
+          const unsigned char *, size_t);
+
+#endif /* __KERNEL__ */
+
+#endif
diff -X dontdiff -Nru linux-2.6.12-rc3.orig/lib/textsearch.c
linux-2.6.12-rc3/lib/textsearch.c
--- linux-2.6.12-rc3.orig/lib/textsearch.c 1970-01-01 01:00:00.000000000 +0100
+++ linux-2.6.12-rc3/lib/textsearch.c 2005-05-04 23:01:42.000000000 +0200
@@ -0,0 +1,241 @@
+/*
+ * lib/textsearch.c Generic text search interface
+ *
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License
+ *  as published by the Free Software Foundation; either version
+ *  2 of the License, or (at your option) any later version.
+ *
+ * Authors: Thomas Graf <tgraf@suug.ch>
+ *
+ * ==========================================================================
+ *
+ * The textsearch infrastructure provides text searching facitilies for both
+ * linear and non-linear data. Unlike many other
+ *
+ * Before a textsearch can be performed, a configuration must be created
+ * by calling textsearch_prepare() specyfing the searching algorithm and
+ * the pattern to match. The returned configuration may then be used for
+ * an arbitary amount of times and even in parallel as long as you
+ * provide a separate struct ts_state variable for every instance.
+ *
+ * The actual search is performed by either calling textsearch_find_text()
+ * for linear text or by providing an own get_text() implementation and
+ * calling textsearch_find(). Both functions will returns the position
+ * of the first matching occurrence of the patern. Subsequent matches
+ * may be retrived via textsearch_next() irrespective of the linearity
+ * of the text.
+ *
+ * Once you're done using a configuration you must give back the
+ * allocted resources by calling textsearch_put().
+ *
+ * Example:
+ *   int pos;
+ *   struct ts_config *conf;
+ *   struct ts_state state;
+ *   conar char *pattern = "chicken";
+ *   const char *example = "We dance the funky chicken";
+ *
+ *   conf = textsearch_prepare("kmp", pattern, strlen(pattern), GFP_KERNEL,
1);
+ *   if (IS_ERR(conf)) {
```

```
+ *       err = PTR_ERR(conf);
+ *       goto errout;
+ *   }
+ *
+ *   pos = textsearch_find_text(conf, &state, example, strlen(example));
+ *   if (pos >= 0)
+ *       panic("Oh my god, dancing chickens at %d\n", pos);
+ *
+ *   textsearch_put(conf);
+ *   kfree(conf);
+ *
+ * ==========================================================================
+ */
+
+#include <linux/config.h>
+#include <linux/module.h>
+#include <linux/types.h>
+#include <linux/string.h>
+#include <linux/err.h>
+#include <linux/textsearch.h>
+
+static LIST_HEAD(ts_ops);
+static DEFINE_RWLOCK(ts_mod_lock);
+
+static inline struct ts_ops *lookup_ts_algo(const char *name)
+{
+ struct ts_ops *o;
+
+ read_lock(&ts_mod_lock);
+ list_for_each_entry(o, &ts_ops, list) {
+  if (!strcmp(name, o->name)) {
+   if (!try_module_get(o->owner))
+    o = NULL;
+   read_unlock(&ts_mod_lock);
+   return o;
+  }
+ }
+ read_unlock(&ts_mod_lock);
+
+ return NULL;
+}
+
+/**
+ * textsearch_register - register a textsearch module
+ * @ops: operations lookup table
+ *
+ * This function must be called by textsearch modules to announce
+ * their presence. The given @ops must have %name set to a unique
+ * identifier and the callbacks find() and init() must be implemented.
+ *
+ * Returns -EEXISTS if another module has already registered with
+ *         same name.
+ */
+int textsearch_register(struct ts_ops *ops)
+{
+ int err = -EEXIST;
+ struct ts_ops *o;
```

```
+
+ if (ops->name == NULL || ops->find == NULL || ops->init == NULL)
+  return -EINVAL;
+
+ write_lock(&ts_mod_lock);
+ list_for_each_entry(o, &ts_ops, list)
+  if (!strcmp(ops->name, o->name))
+   goto errout;
+
+ list_add_tail(&ops->list, &ts_ops);
+ err = 0;
+errout:
+ write_unlock(&ts_mod_lock);
+ return err;
+}
+
+/**
+ * textsearch_unregister - unregister a textsearch module
+ * @ops: operations lookup table
+ *
+ * This function must be called by textsearch modules to announce
+ * their disappearance for examples when the module gets unloaded.
+ * The @ops parameter must be the same as the one during the
+ * registration.
+ *
+ * Returns -ENOENT if no matching textsearch registration was found.
+ */
+int textsearch_unregister(struct ts_ops *ops)
+{
+ int err = 0;
+ struct ts_ops *o;
+
+ write_lock(&ts_mod_lock);
+ list_for_each_entry(o, &ts_ops, list) {
+  if (o == ops) {
+   list_del(&o->list);
+   goto out;
+  }
+ }
+
+ err = -ENOENT;
+out:
+ write_unlock(&ts_mod_lock);
+ return err;
+}
+
+static inline int get_linear_text(int offset, unsigned char **text,
+      struct ts_config *conf,
+      struct ts_state *state)
+{
+ unsigned char *string = (unsigned char *) state->args[0];
+ size_t len = state->args[1];
+
+ if (offset < len) {
+  *text = string + offset;
+  return len - offset;
+ } else
+  return 0;
+}
+
+/**
+ * textsearch_find_continuous - search a pattern in continuous/linear text
+ * @conf: search configuration
+ * @state: search state
+ * @text: text to search in
+ * @len: length of text
+ *
+ * A simplified version of textsearch_find() for continuous/linear text.
+ * Call textsearch_next() to retrieve subsequent matches.
+ *
+ * Returns the position of first occurrence of the pattern or a
+ *         negative number if no match was found.
+ */
+int textsearch_find_continuous(struct ts_config *conf, struct ts_state
*state,
+    const unsigned char *text, size_t len)
+{
+ conf->get_text = get_linear_text;
+ state->args[0] = (long) text;
+ state->args[1] = len;
+
+ return textsearch_find(conf, state);
+}
+
+/**
+ * textsearch_prepare - Prepare a text search
+ * @algo: name of search algorithm
+ * @pattern: pattern data
+ * @len: length of pattern
+ * @gfp_mask: allocation mask
+ * @autoload: bool indicating whether to autoload modules
+ *
+ * Looks up the search algorithm module and creates a new textsearch
+ * configuration for the specified pattern. Upon completion all
+ * necessary refcnts are held and the configuration must be put back
+ * using textsearch_put() after usage.
+ *
+ * Note: The format of the pattern may not be compatible between
+ *       the various search algorithms.
+ *
+ * Returns a new textsearch configuration according to the specified
+ *         parameters or a ERR_PTR().
+ */
+struct ts_config *textsearch_prepare(const char *algo,
+         const unsigned char *pattern, size_t len,
+         int gfp_mask, int autoload)
+{
+ int err = -ENOENT;
+ struct ts_config *conf;
+ struct ts_ops *ops;
+
+ ops = lookup_ts_algo(algo);
+#ifdef CONFIG_KMOD
```

```
+ /* Why not always autoload you may ask. Some users may be
+  * in a situation where requesting a module may deadlock,
+  * especially when the module is located on a NFS mount. */
+ if (ops == NULL && autoload) {
+  request_module("ts_%s", algo);
+  ops = lookup_ts_algo(algo);
+ }
+#endif
+
+ if (ops == NULL)
+  goto errout;
+
+ conf = ops->init(pattern, len, gfp_mask);
+ if (IS_ERR(conf)) {
+  err = PTR_ERR(conf);
+  goto errout;
+ }
+
+ conf->ops = ops;
+ return conf;
+
+errout:
+ if (ops)
+  module_put(ops->owner);
+
+ return ERR_PTR(err);
+}
+
+EXPORT_SYMBOL(textsearch_register);
+EXPORT_SYMBOL(textsearch_unregister);
+EXPORT_SYMBOL(textsearch_prepare);
+EXPORT_SYMBOL(textsearch_find_continuous);
diff -X dontdiff -Nru linux-2.6.12-rc3.orig/lib/ts_kmp.c
linux-2.6.12-rc3/lib/ts_kmp.c
--- linux-2.6.12-rc3.orig/lib/ts_kmp.c 1970-01-01 01:00:00.000000000 +0100
+++ linux-2.6.12-rc3/lib/ts_kmp.c 2005-05-05 00:32:28.000000000 +0200
@@ -0,0 +1,108 @@
+/*
+ * lib/ts_kmp.c  Knuth-Morris-Pratt text search implementation
+ *
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License
+ *  as published by the Free Software Foundation; either version
+ *  2 of the License, or (at your option) any later version.
+ *
+ * Authors: Thomas Graf <tgraf@suug.ch>
+ */
+
+#include <linux/config.h>
+#include <linux/module.h>
+#include <linux/types.h>
+#include <linux/string.h>
+#include <linux/textsearch.h>
+
+#define TS_KMP_PREFIX_TBL(conf) \
+ ((unsigned int *) ((unsigned char *) (conf) \
+      + sizeof(struct ts_config)))
+
+#define TS_KMP_PATTERN(conf) \
+ ((unsigned char *) TS_KMP_PREFIX_TBL(conf) \
+      + (conf->pattern_len * sizeof(unsigned int)))
+
+static int kmp_find(struct ts_config *conf, struct ts_state *state)
+{
+ int i, q = 0, consumed = state->offset;
+ unsigned char *text, *pattern = TS_KMP_PATTERN(conf);
+ unsigned int *prefix_tbl = TS_KMP_PREFIX_TBL(conf);
+ size_t text_len, pattern_len = conf->pattern_len;
+
+ for (;;) {
+  text_len = conf->get_text(consumed, &text, conf, state);
+
+  if (text_len == 0)
+   break;
+
+  for (i = 0; i < text_len; i++) {
+   while (q > 0 && pattern[q] != text[i])
+    q = prefix_tbl[q - 1];
+   if (pattern[q] == text[i])
+    q++;
+   if (q == pattern_len) {
+    state->offset = consumed + i - pattern_len + 1;
+    return state->offset;
+   }
+  }
+
+  consumed += text_len;
+ }
+
+ return -1;
+
+}
+
+static inline void compute_prefix_tbl(const unsigned char *pattern, size_t
len,
+          unsigned int *prefix_tbl)
+{
+ unsigned int k, q;
+
+ for (k = 0, q = 1; q < len; q++) {
+  while (k > 0 && pattern[k] != pattern[q])
+   k = prefix_tbl[k-1];
+  if (pattern[k] == pattern[q])
+   k++;
+  prefix_tbl[q] = k;
+ }
+}
+
+static struct ts_config *kmp_init(const unsigned char *pattern, size_t len,
+      int gfp_mask)
+{
+ struct ts_config *conf;
+
+ conf = alloc_ts_config(len + (len * sizeof(int)), gfp_mask);
+ if (IS_ERR(conf))
+  return conf;
+
+ conf->pattern_len = len;
+ memcpy(TS_KMP_PATTERN(conf), pattern, len);
+ compute_prefix_tbl(pattern, len, TS_KMP_PREFIX_TBL(conf));
+
+ return conf;
+}
+
+static struct ts_ops kmp_ops = {
+ .name = "kmp",
+ .find = kmp_find,
+ .init = kmp_init,
+ .owner = THIS_MODULE,
+ .list = LIST_HEAD_INIT(kmp_ops.list)
+};
+
+static int __init init_kmp(void)
+{
+ return textsearch_register(&kmp_ops);
+}
+
+static void __exit exit_kmp(void)
+{
+ textsearch_unregister(&kmp_ops);
+}
+
+MODULE_LICENSE("GPL");
+
+module_init(init_kmp);
+module_exit(exit_kmp);
diff -X dontdiff -Nru linux-2.6.12-rc3.orig/net/core/skbuff.c
linux-2.6.12-rc3/net/core/skbuff.c
--- linux-2.6.12-rc3.orig/net/core/skbuff.c 2005-04-30 20:22:24.000000000
+0200
+++ linux-2.6.12-rc3/net/core/skbuff.c 2005-05-05 00:35:50.000000000 +0200
@@ -1502,6 +1502,80 @@
   skb_split_no_header(skb, skb1, len, pos);
 }

+static int get_skb_text(int offset, unsigned char **text,
+   struct ts_config *conf, struct ts_state *state)
+{
+ /* args[0]: lower limit
+  * args[1]: upper limit
+  * args[2]: skb
+  * args[3]: fragment index
+  * args[4]: current fragment data buffer
+  * args[5]: octets consumed up to previous fragment */
+ int from = state->args[0], to = state->args[1];
+ struct sk_buff *skb = (struct sk_buff *) state->args[2];
+ int limit = min_t(int, skb_headlen(skb), to);
+ int real_offset = offset + from;
+ skb_frag_t *f;
```

```
+
+ if (!skb_is_nonlinear(skb)) {
+  if (real_offset < limit) {
+linear:
+   *text = skb->data + real_offset;
+   return limit - real_offset;
+  }
+
+  return 0;
+ }
+
+ if (real_offset < limit)
+  goto linear;
+
+next_fragment:
+ f = &skb_shinfo(skb)->frags[state->args[3]];
+ limit = min_t(int, f->size + state->args[5], to);
+
+ if (!state->args[4])
+  state->args[4] = (long) kmap_skb_frag(f);
+
+ if (real_offset < limit) {
+  *text = (unsigned char *) state->args[4] + f->page_offset +
+   (real_offset - (int) state->args[5]);
+  return limit - real_offset;
+ }
+
+ kunmap_skb_frag((void *) state->args[4]);
+ state->args[3]++;
+ state->args[4] = (long) NULL;
+ state->args[5] += f->size;
+
+ if (state->args[3] >= skb_shinfo(skb)->nr_frags)
+  return 0;
+
+ goto next_fragment;
+}
+
+static void get_skb_text_finish(struct ts_config *conf, struct ts_state
*state)
+{
+ if (state->args[4])
+  kunmap_skb_frag((void *) state->args[4]);
+}
+
+int skb_find_text(struct sk_buff *skb, int from, int to,
+    struct ts_config *config, struct ts_state *state)
+{
+ config->get_text = get_skb_text;
+ config->finish = get_skb_text_finish;
+ state->args[0] = from;
+ state->args[1] = to;
+ state->args[2] = (long) skb;
+ state->args[3] = 0;
+ state->args[4] = 0;
+ state->args[5] = skb_headlen(skb);
+
+ return textsearch_find(config, state);
+}
+
+
 void __init skb_init(void)
 {
  skbuff_head_cache = kmem_cache_create("skbuff_head_cache",
@@ -1540,3 +1614,4 @@
 EXPORT_SYMBOL(skb_unlink);
 EXPORT_SYMBOL(skb_append);
 EXPORT_SYMBOL(skb_split);
+EXPORT_SYMBOL(skb_find_text);
diff -X dontdiff -Nru linux-2.6.12-rc3.orig/include/linux/skbuff.h
linux-2.6.12-rc3/include/linux/skbuff.h
--- linux-2.6.12-rc3.orig/include/linux/skbuff.h 2005-04-30 20:22:19.000000000
+0200
+++ linux-2.6.12-rc3/include/linux/skbuff.h 2005-05-05 00:12:46.000000000
+0200
@@ -28,6 +28,7 @@
 #include <linux/poll.h>
 #include <linux/net.h>
 #include <net/checksum.h>
+#include <linux/textsearch.h>

 #define HAVE_ALLOC_SKB  /* For the drivers to know */
 #define HAVE_ALIGNABLE_SKB /* Ditto 8)     */
@@ -1186,6 +1187,9 @@
 extern void        skb_split(struct sk_buff *skb,
      struct sk_buff *skb1, const u32 len);

+extern int skb_find_text(struct sk_buff *skb, int from, int to,
+    struct ts_config *config, struct ts_state *state);
+
 static inline void *skb_header_pointer(const struct sk_buff *skb, int offset,
            int len, void *buffer)
 {
```