```
#|

 Copyright (C) 1994 by Yuan Yu.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Yuan Yu PROVIDES ABSOLUTELY NO WARRANTY.  THE EVENT SCRIPT IS PROVIDED "AS IS"
 WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT
 LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
 SCRIPT IS WITH YOU.  SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF
 ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

 IN NO EVENT WILL Yuan Yu BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS,
 LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT
 OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS
 OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD
 PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR
 FOR ANY CLAIM BY ANY OTHER PARTY.

|#
```

EVENT: Start with the library "mc20-2" using the compiled version.

```
;          Proof of the Correctness of the STRNCPY Function
#|
This is part of our effort to verify the Berkeley string library.  The
Berkeley string library is widely used as part of the Berkeley Unix OS.

This is the source code of strncpy function in the Berkeley string library.

char *
strncpy(dst, src, n)
char *dst;
const char *src;
register size_t n;
{
if (n != 0) {
register char *d = dst;
register const char *s = src;
```

```
do {
if ((*d++ = *s++) == 0) {
/* NUL pad the remaining n-1 bytes */
while (--n != 0)
*d++ = 0;
break;
}
} while (--n != 0);
}
return (dst);
}
```

The MC68020 assembly code of the C function strncpy on SUN-3 is given as follows. This binary is generated by "gcc -O".

```
0x2650 <strncpy>:       linkw fp,#0
0x2654 <strncpy+4>:     movel d2,sp@-
0x2656 <strncpy+6>:     movel fp@(8),d2
0x265a <strncpy+10>:    movel fp@(16),d1
0x265e <strncpy+14>:    beq 0x267a <strncpy+42>
0x2660 <strncpy+16>:    moveal d2,a0
0x2662 <strncpy+18>:    moveal fp@(12),a1
0x2666 <strncpy+22>:    moveb a1@+,d0
0x2668 <strncpy+24>:    moveb d0,a0@+
0x266a <strncpy+26>:    bne 0x2676 <strncpy+38>
0x266c <strncpy+28>:    bra 0x2670 <strncpy+32>
0x266e <strncpy+30>:    clrb a0@+
0x2670 <strncpy+32>:    subl #1,d1
0x2672 <strncpy+34>:    bne 0x266e <strncpy+30>
0x2674 <strncpy+36>:    bra 0x267a <strncpy+42>
0x2676 <strncpy+38>:    subl #1,d1
0x2678 <strncpy+40>:    bne 0x2666 <strncpy+22>
0x267a <strncpy+42>:    movel d2,d0
0x267c <strncpy+44>:    movel fp@(-4),d2
0x2680 <strncpy+48>:    unlk fp
0x2682 <strncpy+50>:    rts
```

The machine code of the above program is:

```
<strncpy>:     0x4e56  0x0000  0x2f02  0x242e  0x0008  0x222e  0x0010  0x671a
<strncpy+16>:  0x2042  0x226e  0x000c  0x1019  0x10c0  0x660a  0x6002  0x4218
<strncpy+32>:  0x5381  0x66fa  0x6004  0x5381  0x66ec  0x2002  0x242e  0xfffc
<strncpy+48>:  0x4e5e  0x4e75
```

```
'(78      86      0       0       47      2       36      46
  0       8       34      46      0       16      103     26
  32      66      34      110     0       12      16      25
  16      192     102     10      96      2       66      24
  83      129     102     250     96      4       83      129
  102     236     32      2       36      46      255     252
  78      94      78      117)
|#
; in the logic, the above program is defined by (strncpy-code).
```

DEFINITION:
STRNCPY-CODE
= `'(78 86 0 0 47 2 36 46 0 8 34 46 0 16 103 26 32 66 34 110 0 12 16 25 16 192 102 10 96 2 66 24 83 129 102 250 96 4 83 129 102 236 32 2 36 46 255 252 78 94 78 117)`

`; the computation time of the program.`

DEFINITION:
strncpy-t0 $(i,\ n)$
= **if** $(n - 1) = 0$ **then** 7
else splus (3, strncpy-t0 $(1 + i,\ n - 1)$) **endif**

DEFINITION: strncpy-t1 $(i,\ n)$ = splus (4, strncpy-t0 $(1 + i,\ n)$)

DEFINITION:
strncpy-t2 $(i,\ n,\ lst2)$
= **if** get-nth $(i,\ lst2) = 0$ **then** strncpy-t1 $(i,\ n)$
elseif $(n - 1) = 0$ **then** 9
else splus (5, strncpy-t2 $(1 + i,\ n - 1,\ lst2)$) **endif**

DEFINITION:
strncpy-t $(n,\ lst2)$
= **if** $n \simeq 0$ **then** 9
else splus (7, strncpy-t2 $(0,\ n,\ lst2)$) **endif**

`; two induction hints.`

DEFINITION:
strncpy-induct1 $(s,\ i*,\ i,\ n,\ lst1)$
= **if** $(n - 1) = 0$ **then t**
else strncpy-induct1 (stepn $(s,\ 3)$,
add $(32,\ i*,\ 1)$,
$1 + i$,
$n - 1$,
put-nth $(0,\ i,\ lst1)$) **endif**

DEFINITION:
strncpy-induct2 (s, $i*$, i, n, *lst1*, *lst2*)
= **if** get-nth (i, *lst2*) = 0 **then t**
elseif ($n - 1$) = 0 **then t**
else strncpy-induct2 (stepn (s, 5),
add (32, $i*$, 1),
$1 + i$,
$n - 1$,
put-nth (get-nth (i, *lst2*), i, *lst1*),
lst2) **endif**

```
; the preconditions of the initial state.
```

DEFINITION:
strncpy-statep (s, *str1*, n, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (mc-pc (s), mc-mem (s), 52)
∧ mcode-addrp (mc-pc (s), mc-mem (s), STRNCPY-CODE)
∧ ram-addrp (sub (32, 8, read-sp (s)), mc-mem (s), 24)
∧ ram-addrp (*str1*, mc-mem (s), n)
∧ mem-lst (1, *str1*, mc-mem (s), n, *lst1*)
∧ ram-addrp (*str2*, mc-mem (s), *n2*)
∧ mem-lst (1, *str2*, mc-mem (s), *n2*, *lst2*)
∧ disjoint (sub (32, 8, read-sp (s)), 24, *str1*, n)
∧ disjoint (sub (32, 8, read-sp (s)), 24, *str2*, *n2*)
∧ disjoint (*str1*, n, *str2*, *n2*)
∧ (slen (0, n, *lst2*) < *n2*)
∧ (*str1* = read-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
∧ (*str2* = read-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
∧ (n = uread-mem (add (32, read-sp (s), 12), mc-mem (s), 4))
∧ (*n2* ∈ **N**))

```
; an intermediate state s0.
```

DEFINITION:
strncpy-s0p (s, $i*$, i, *str1*, $n_$, *lst1*, *str2*, *n2*, *lst2*, n)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (sub (32, 22, mc-pc (s)), mc-mem (s), 52)
∧ mcode-addrp (sub (32, 22, mc-pc (s)), mc-mem (s), STRNCPY-CODE)
∧ ram-addrp (sub (32, 4, read-an (32, 6, s)), mc-mem (s), 24)
∧ ram-addrp (*str1*, mc-mem (s), $n_$)
∧ mem-lst (1, *str1*, mc-mem (s), $n_$, *lst1*)
∧ ram-addrp (*str2*, mc-mem (s), *n2*)

$\wedge$ mem-lst (1, *str2*, mc-mem (*s*), *n2*, *lst2*)
$\wedge$ disjoint (sub (32, 4, read-an (32, 6, *s*)), 24, *str1*, *n_*)
$\wedge$ disjoint (sub (32, 4, read-an (32, 6, *s*)), 24, *str2*, *n2*)
$\wedge$ disjoint (*str1*, *n_*, *str2*, *n2*)
$\wedge$ (slen (*i*, *n_*, *lst2*) $<$ *n2*)
$\wedge$ equal* (read-an (32, 0, *s*), add (32, *str1*, *i**))
$\wedge$ equal* (read-an (32, 1, *s*), add (32, *str2*, *i**))
$\wedge$ (*str1* $=$ read-dn (32, 2, *s*))
$\wedge$ (*n* $=$ nat-to-uint (read-dn (32, 1, *s*)))
$\wedge$ ((*i* $+$ *n*) $\leq$ *n_*)
$\wedge$ (*n* $\neq$ 0)
$\wedge$ (*i** $\in$ **N**)
$\wedge$ nat-rangep (*i**, 32)
$\wedge$ (*i* $=$ nat-to-uint (*i**))
$\wedge$ (*n2* $\in$ **N**)
$\wedge$ (*n_* $\in$ **N**)
$\wedge$ uint-rangep (*n_*, 32))

```
; an intermediate state s1.
```

DEFINITION:
strncpy-s1p (*s*, *i**, *i*, *str1*, *n_*, *lst1*, *str2*, *n2*, *lst2*, *n*)
$=$ ((mc-status (*s*) $=$ 'running)
$\wedge$ evenp (mc-pc (*s*))
$\wedge$ rom-addrp (sub (32, 32, mc-pc (*s*)), mc-mem (*s*), 52)
$\wedge$ mcode-addrp (sub (32, 32, mc-pc (*s*)), mc-mem (*s*), STRNCPY-CODE)
$\wedge$ ram-addrp (sub (32, 4, read-an (32, 6, *s*)), mc-mem (*s*), 24)
$\wedge$ ram-addrp (*str1*, mc-mem (*s*), *n_*)
$\wedge$ mem-lst (1, *str1*, mc-mem (*s*), *n_*, *lst1*)
$\wedge$ ram-addrp (*str2*, mc-mem (*s*), *n2*)
$\wedge$ mem-lst (1, *str2*, mc-mem (*s*), *n2*, *lst2*)
$\wedge$ disjoint (sub (32, 4, read-an (32, 6, *s*)), 24, *str1*, *n_*)
$\wedge$ disjoint (sub (32, 4, read-an (32, 6, *s*)), 24, *str2*, *n2*)
$\wedge$ disjoint (*str1*, *n_*, *str2*, *n2*)
$\wedge$ equal* (read-an (32, 0, *s*), add (32, *str1*, *i**))
$\wedge$ (*str1* $=$ read-dn (32, 2, *s*))
$\wedge$ (*n* $=$ nat-to-uint (read-dn (32, 1, *s*)))
$\wedge$ (*i* $=$ nat-to-uint (*i**))
$\wedge$ ((*i* $+$ (*n* $-$ 1)) $\leq$ *n_*)
$\wedge$ (*n* $\neq$ 0)
$\wedge$ (*i** $\in$ **N**)
$\wedge$ nat-rangep (*i**, 32)
$\wedge$ (*n2* $\in$ **N**)
$\wedge$ (*n_* $\in$ **N**)
$\wedge$ uint-rangep (*n_*, 32))

```
; from the initial state s to exit: s --> sn, when n = 0.
```

THEOREM: strncpy-s-sn
(strncpy-statep (s, *str1*, n, *lst1*, *str2*, *n2*, *lst2*) $\wedge$ ($n \simeq 0$))
$\rightarrow$ ((mc-status (stepn (s, 9)) = 'running)
$\wedge$ (mc-pc (stepn (s, 9)) = rts-addr (s))
$\wedge$ (read-dn (32, 0, stepn (s, 9)) = *str1*)
$\wedge$ mem-lst (1, *str1*, mc-mem (stepn (s, 9)), n, *lst1*)
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 9)))
= add (32, read-an (32, 7, s), 4))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 9))) = read-an (32, 6, s)))

THEOREM: strncpy-s-sn-rfile
(strncpy-statep (s, *str1*, n, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ ($n \simeq 0$)
$\wedge$ (*oplen* $\leq$ 32)
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 9)))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: strncpy-s-sn-mem
(strncpy-statep (s, *str1*, n, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ ($n \simeq 0$)
$\wedge$ disjoint (x, k, sub (32, 8, read-sp (s)), 24))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 9)), k) = read-mem (x, mc-mem (s), k))

```
; from the initial state s to s0: s --> s0.
```

THEOREM: strncpy-s-s0
(strncpy-statep (s, *str1*, n, *lst1*, *str2*, *n2*, *lst2*) $\wedge$ ($n \not\simeq 0$))
$\rightarrow$ strncpy-s0p (stepn (s, 7), 0, 0, *str1*, n, *lst1*, *str2*, *n2*, *lst2*, n)

THEOREM: strncpy-s-s0-else
(strncpy-statep (s, *str1*, n, *lst1*, *str2*, *n2*, *lst2*) $\wedge$ ($n \not\simeq 0$))
$\rightarrow$ ((linked-rts-addr (stepn (s, 7)) = rts-addr (s))
$\wedge$ (linked-a6 (stepn (s, 7)) = read-an (32, 6, s))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 7)))
= sub (32, 4, read-sp (s)))
$\wedge$ (rn-saved (stepn (s, 7)) = read-rn (32, 2, mc-rfile (s))))

THEOREM: strncpy-s-s0-rfile
(strncpy-statep (s, *str1*, n, *lst1*, *str2*, *n2*, *lst2*) $\wedge$ ($n \not\simeq 0$) $\wedge$ d3-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 7)))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: strncpy-s-s0-mem
(strncpy-statep (s, *str1*, n, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ ($n \not\simeq 0$)
$\wedge$ disjoint (x, k, sub (32, 8, read-sp (s)), 24))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 7)), k) = read-mem (x, mc-mem (s), k))

```
; from s0 to exit: s0 --> sn.  By induction.
; base case 1:  s0 --> sn, when lst1[i] = 0.
; s0 --> s1.
```

THEOREM: strncpy-s0-s1
(strncpy-s0p (s, i*, i, *str1*, $n_$, *lst1*, *str2*, *n2*, *lst2*, n)
$\wedge$ (get-nth (i, *lst2*) = 0))
$\rightarrow$ (strncpy-s1p (stepn (s, 4),
add (32, i*, 1),
$1 + i$,
str1,
$n_$,
put-nth (get-nth (i, *lst2*), i, *lst1*),
str2,
n2,
lst2,
n)
$\wedge$ (linked-rts-addr (stepn (s, 4)) = linked-rts-addr (s))
$\wedge$ (linked-a6 (stepn (s, 4)) = linked-a6 (s))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 4)))
= read-rn (32, 14, mc-rfile (s)))
$\wedge$ (rn-saved (stepn (s, 4)) = rn-saved (s)))

THEOREM: strncpy-s0-s1-rfile
(strncpy-s0p (s, i*, i, *str1*, $n_$, *lst1*, *str2*, *n2*, *lst2*, n)
$\wedge$ (get-nth (i, *lst2*) = 0)
$\wedge$ d3-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 4)))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: strncpy-s0-s1-mem
(strncpy-s0p (s, i*, i, *str1*, $n_$, *lst1*, *str2*, *n2*, *lst2*, n)
$\wedge$ (get-nth (i, *lst2*) = 0)
$\wedge$ disjoint (x, k, *str1*, $n_$))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 4)), k) = read-mem (x, mc-mem (s), k))

```
; s1 --> sn.  By induction.
; base case: s1 --> exit, when n-1 = 0.
```

THEOREM: strncpy-s1-sn-base
(strncpy-s1p (s, i^*, i, *str1*, $n_$, *lst1*, *str2*, *n2*, *lst2*, n) $\wedge$ (($n - 1$) = 0))
$\rightarrow$ ((mc-status (stepn (s, 7)) = 'running)
$\wedge$ (mc-pc (stepn (s, 7)) = linked-rts-addr (s))
$\wedge$ (read-dn (32, 0, stepn (s, 7)) = *str1*)
$\wedge$ mem-lst (1, *str1*, mc-mem (stepn (s, 7)), $n_$, *lst1*)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 7))) = linked-a6 (s))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 7)))
= add (32, read-an (32, 6, s), 8))
$\wedge$ (read-mem (x, mc-mem (stepn (s, 7)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: strncpy-s1-sn-rfile-base
(strncpy-s1p (s, i^*, i, *str1*, $n_$, *lst1*, *str2*, *n2*, *lst2*, n)
$\wedge$ (($n - 1$) = 0)
$\wedge$ (*oplen* $\leq$ 32)
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 7)))
= **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (s))
else head (rn-saved (s), *oplen*) **endif**)

```
; induction case: s1 --> s1, when n-1 =\= 0.
```

THEOREM: strncpy-s1-s1
(strncpy-s1p (s, i^*, i, *str1*, $n_$, *lst1*, *str2*, *n2*, *lst2*, n) $\wedge$ (($n - 1$) $\neq$ 0))
$\rightarrow$ (strncpy-s1p (stepn (s, 3),
add (32, i^*, 1),
$1 + i$,
str1,
$n_$,
put-nth (0, i, *lst1*),
str2,
n2,
lst2,
$n - 1$)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 3)))
= read-rn (32, 14, mc-rfile (s)))
$\wedge$ (linked-a6 (stepn (s, 3)) = linked-a6 (s))
$\wedge$ (linked-rts-addr (stepn (s, 3)) = linked-rts-addr (s))
$\wedge$ (rn-saved (stepn (s, 3)) = rn-saved (s)))

THEOREM: strncpy-s1-s1-rfile
(strncpy-s1p (s, i^*, i, *str1*, $n_$, *lst1*, *str2*, *n2*, *lst2*, n)
$\wedge$ (($n - 1$) $\neq$ 0)
$\wedge$ d3-7a2-5p (*rn*))

→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 3)))
  = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: strncpy-s1-s1-mem
(strncpy-s1p (*s*, *i**, *i*, *str1*, *n_*, *lst1*, *str2*, *n2*, *lst2*, *n*)
∧ disjoint (*x*, *k*, *str1*, *n_*)
∧ ((*n* − 1) ≠ 0))
→ (read-mem (*x*, mc-mem (stepn (*s*, 3)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; put together. s1 --> exit.
```

THEOREM: strncpy-s1-sn
let *sn* **be** stepn (*s*, strncpy-t0 (*i*, *n*))
in
strncpy-s1p (*s*, *i**, *i*, *str1*, *n_*, *lst1*, *str2*, *n2*, *lst2*, *n*)
→ ((mc-status (*sn*) = `'running`)
  ∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
  ∧ (read-dn (32, 0, *sn*) = *str1*)
  ∧ mem-lst (1, *str1*, mc-mem (*sn*), *n_*, zero-list1 (*i*, *n*, *lst1*))
  ∧ (read-rn (32, 14, mc-rfile (*sn*)) = linked-a6 (*s*))
  ∧ (read-rn (32, 15, mc-rfile (*sn*))
    = add (32, read-an (32, 6, *s*), 8))) **endlet**

THEOREM: strncpy-s1-sn-rfile
(strncpy-s1p (*s*, *i**, *i*, *str1*, *n_*, *lst1*, *str2*, *n2*, *lst2*, *n*)
∧ (*oplen* ≤ 32)
∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strncpy-t0 (*i*, *n*))))
  = **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
    else head (rn-saved (*s*), *oplen*) **endif**)

THEOREM: strncpy-s1-sn-mem
(strncpy-s1p (*s*, *i**, *i*, *str1*, *n_*, *lst1*, *str2*, *n2*, *lst2*, *n*) ∧ disjoint (*x*, *k*, *str1*, *n_*))
→ (read-mem (*x*, mc-mem (stepn (*s*, strncpy-t0 (*i*, *n*))), *k*)
  = read-mem (*x*, mc-mem (*s*), *k*))

```
; put together (base case 1). s0 --> exit.
```

THEOREM: strncpy-s0-sn-base1
let *sn* **be** stepn (*s*, strncpy-t1 (*i*, *n*))
in
(strncpy-s0p (*s*, *i**, *i*, *str1*, *n_*, *lst1*, *str2*, *n2*, *lst2*, *n*)
∧ (get-nth (*i*, *lst2*) = 0))
→ ((mc-status (*sn*) = `'running`)
  ∧ (mc-pc (*sn*) = linked-rts-addr (*s*))

```
    ∧   (read-dn (32, 0, sn) = str1)
    ∧   mem-lst (1, str1, mc-mem (sn), n_, zero-list (i, n, lst1))
    ∧   (read-rn (32, 14, mc-rfile (sn)) = linked-a6 (s))
    ∧   (read-rn (32, 15, mc-rfile (sn))
          =   add (32, read-an (32, 6, s), 8))) endlet
```

THEOREM: strncpy-s0-sn-rfile-base1

```
 (strncpy-s0p (s, i*, i, str1, n_, lst1, str2, n2, lst2, n)
  ∧   (get-nth (i, lst2) = 0)
  ∧   (oplen ≤ 32)
  ∧   d2-7a2-5p (rn))
  →   (read-rn (oplen, rn, mc-rfile (stepn (s, strncpy-t1 (i, n))))
        =   if d3-7a2-5p (rn) then read-rn (oplen, rn, mc-rfile (s))
            else head (rn-saved (s), oplen) endif)
```

THEOREM: strncpy-s0-sn-mem-base1

```
 (strncpy-s0p (s, i*, i, str1, n_, lst1, str2, n2, lst2, n)
  ∧   (get-nth (i, lst2) = 0)
  ∧   disjoint (x, k, str1, n_))
  →   (read-mem (x, mc-mem (stepn (s, strncpy-t1 (i, n))), k)
        =   read-mem (x, mc-mem (s), k))
```

```
; base case 2: s0 --> sn, when lst2[i] =\= 0, n-1 = 0.
```

THEOREM: strncpy-s0-sn-base2

```
 (strncpy-s0p (s, i*, i, str1, n_, lst1, str2, n2, lst2, n)
  ∧   (get-nth (i, lst2) ≠ 0)
  ∧   ((n − 1) = 0))
  →   ((mc-status (stepn (s, 9)) = 'running)
        ∧   (mc-pc (stepn (s, 9)) = linked-rts-addr (s))
        ∧   (read-dn (32, 0, stepn (s, 9)) = str1)
        ∧   mem-lst (1,
                     str1,
                     mc-mem (stepn (s, 9)),
                     n_,
                     put-nth (get-nth (i, lst2), i, lst1))
        ∧   (read-rn (32, 14, mc-rfile (stepn (s, 9))) = linked-a6 (s))
        ∧   (read-rn (32, 15, mc-rfile (stepn (s, 9)))
              =   add (32, read-an (32, 6, s), 8)))
```

THEOREM: strncpy-s0-sn-rfile-base2

```
 (strncpy-s0p (s, i*, i, str1, n_, lst1, str2, n2, lst2, n)
  ∧   (get-nth (i, lst2) ≠ 0)
  ∧   ((n − 1) = 0)
  ∧   (oplen ≤ 32)
```

$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 9)))
$=$ **if** d3-7a2-5p (rn) **then** read-rn ($oplen$, rn, mc-rfile (s))
else head (rn-saved (s), $oplen$) **endif**)

THEOREM: strncpy-s0-sn-mem-base2
(strncpy-s0p (s, $i*$, i, $str1$, $n_$, $lst1$, $str2$, $n2$, $lst2$, n)
$\wedge$ disjoint (x, k, $str1$, $n_$)
$\wedge$ (get-nth (i, $lst2$) $\neq$ 0)
$\wedge$ (($n - 1$) $=$ 0))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 9)), k) $=$ read-mem (x, mc-mem (s), k))

```
; induction case:  s0 --> s0, when lst[i] =\= 0, n-1 =\= 0.
```

THEOREM: strncpy-s0-s0
(strncpy-s0p (s, $i*$, i, $str1$, $n_$, $lst1$, $str2$, $n2$, $lst2$, n)
$\wedge$ (get-nth (i, $lst2$) $\neq$ 0)
$\wedge$ (($n - 1$) $\neq$ 0))
$\rightarrow$ (strncpy-s0p (stepn (s, 5),
add (32, $i*$, 1),
$1 + i$,
$str1$,
$n_$,
put-nth (get-nth (i, $lst2$), i, $lst1$),
$str2$,
$n2$,
$lst2$,
$n - 1$)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 5)))
$=$ read-rn (32, 14, mc-rfile (s)))
$\wedge$ (linked-a6 (stepn (s, 5)) $=$ linked-a6 (s))
$\wedge$ (linked-rts-addr (stepn (s, 5)) $=$ linked-rts-addr (s))
$\wedge$ (rn-saved (stepn (s, 5)) $=$ rn-saved (s)))

THEOREM: strncpy-s0-s0-rfile
(strncpy-s0p (s, $i*$, i, $str1$, $n_$, $lst1$, $str2$, $n2$, $lst2$, n)
$\wedge$ (get-nth (i, $lst2$) $\neq$ 0)
$\wedge$ (($n - 1$) $\neq$ 0)
$\wedge$ d3-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 5)))
$=$ read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strncpy-s0-s0-mem
(strncpy-s0p (s, $i*$, i, $str1$, $n_$, $lst1$, $str2$, $n2$, $lst2$, n)
$\wedge$ disjoint (x, k, $str1$, $n_$)

```
 ∧   (get-nth (i, lst2) ≠ 0)
 ∧   ((n − 1) ≠ 0))
 →   (read-mem (x, mc-mem (stepn (s, 5)), k) = read-mem (x, mc-mem (s), k))

; put together. s0 --> exit.

THEOREM: strncpy-s0-sn
 let sn be stepn (s, strncpy-t2 (i, n, lst2))
 in
 strncpy-s0p (s, i*, i, str1, n_, lst1, str2, n2, lst2, n)
 →   ((mc-status (sn) = 'running)
      ∧   (mc-pc (sn) = linked-rts-addr (s))
      ∧   (read-dn (32, 0, sn) = str1)
      ∧   mem-lst (1, str1, mc-mem (sn), n_, strncpy1 (i, n, lst1, lst2))
      ∧   (read-rn (32, 14, mc-rfile (sn)) = linked-a6 (s))
      ∧   (read-rn (32, 15, mc-rfile (sn))
            =   add (32, read-an (32, 6, s), 8))) endlet

THEOREM: strncpy-s0-sn-rfile
 (strncpy-s0p (s, i*, i, str1, n_, lst1, str2, n2, lst2, n)
 ∧   d2-7a2-5p (rn)
 ∧   (oplen ≤ 32))
 →   (read-rn (oplen, rn, mc-rfile (stepn (s, strncpy-t2 (i, n, lst2))))
      =   if d3-7a2-5p (rn) then read-rn (oplen, rn, mc-rfile (s))
          else head (rn-saved (s), oplen) endif)

THEOREM: strncpy-s0-sn-mem
 (strncpy-s0p (s, i*, i, str1, n_, lst1, str2, n2, lst2, n) ∧ disjoint (x, k, str1, n_))
 →   (read-mem (x, mc-mem (stepn (s, strncpy-t2 (i, n, lst2))), k)
      =   read-mem (x, mc-mem (s), k))

; the correctness of strncpy.

THEOREM: strncpy-correctness
 let sn be stepn (s, strncpy-t (n, lst2))
 in
 strncpy-statep (s, str1, n, lst1, str2, n2, lst2)
 →   ((mc-status (sn) = 'running)
      ∧   (mc-pc (sn) = rts-addr (s))
      ∧   (read-rn (32, 14, mc-rfile (sn))
            =   read-rn (32, 14, mc-rfile (s)))
      ∧   (read-rn (32, 15, mc-rfile (sn))
            =   add (32, read-an (32, 7, s), 4))
      ∧   ((d2-7a2-5p (rn) ∧ (oplen ≤ 32))
            →   (read-rn (oplen, rn, mc-rfile (sn))
```

$$
\begin{array}{lll}
 & & = \quad \text{read-rn}\,(\mathit{oplen},\ \mathit{rn},\ \text{mc-rfile}\,(s)))) \\
\wedge & & ((\text{disjoint}\,(x,\ k,\ \mathit{str1},\ n) \\
 & & \quad \wedge \quad \text{disjoint}\,(x,\ k,\ \text{sub}\,(\mathtt{32},\ \mathtt{8},\ \text{read-sp}\,(s)),\ \mathtt{24})) \\
 & & \quad \rightarrow \quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(\mathit{sn}),\ k) \\
 & & \qquad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k))) \\
\wedge & & (\text{read-dn}\,(\mathtt{32},\ \mathtt{0},\ \mathit{sn}) = \mathit{str1}) \\
\wedge & & \text{mem-lst}\,(\mathtt{1},\ \mathit{str1},\ \text{mc-mem}\,(\mathit{sn}),\ n,\ \text{strncpy}\,(n,\ \mathit{lst1},\ \mathit{lst2}))) \ \textbf{endlet}
\end{array}
$$

EVENT: Disable strncpy-t.

```
; some properties of strncpy.
; see file cstring.events.
```

Index

add, 3–12

d2-7a2-5p, 6, 8–12
d3-7a2-5p, 6–12
disjoint, 4–7, 9–13

equal*, 5
evenp, 4, 5

get-nth, 3, 4, 7, 9–12

head, 8–12

linked-a6, 6–12
linked-rts-addr, 6–12

mc-mem, 4–13
mc-pc, 4–6, 8–10, 12
mc-rfile, 6–13
mc-status, 4–6, 8–10, 12
mcode-addrp, 4, 5
mem-lst, 4–6, 8–10, 12, 13

nat-rangep, 5
nat-to-uint, 5

put-nth, 3, 4, 7, 8, 10, 11

ram-addrp, 4, 5
read-an, 4–6, 8–10, 12
read-dn, 5, 6, 8–10, 12, 13
read-mem, 4, 6–13
read-rn, 6–13
read-sp, 4, 6, 7, 13
rn-saved, 6–12
rom-addrp, 4, 5
rts-addr, 6, 12

slen, 4, 5
splus, 3
stepn, 3, 4, 6–12
strncpy, 13
strncpy-code, 3–5
strncpy-correctness, 12
strncpy-induct1, 3
strncpy-induct2, 4
strncpy-s-s0, 6
strncpy-s-s0-else, 6
strncpy-s-s0-mem, 7
strncpy-s-s0-rfile, 6
strncpy-s-sn, 6
strncpy-s-sn-mem, 6
strncpy-s-sn-rfile, 6
strncpy-s0-s0, 11
strncpy-s0-s0-mem, 11
strncpy-s0-s0-rfile, 11
strncpy-s0-s1, 7
strncpy-s0-s1-mem, 7
strncpy-s0-s1-rfile, 7
strncpy-s0-sn, 12
strncpy-s0-sn-base1, 9
strncpy-s0-sn-base2, 10
strncpy-s0-sn-mem, 12
strncpy-s0-sn-mem-base1, 10
strncpy-s0-sn-mem-base2, 11
strncpy-s0-sn-rfile, 12
strncpy-s0-sn-rfile-base1, 10
strncpy-s0-sn-rfile-base2, 10
strncpy-s0p, 4, 6, 7, 9–12
strncpy-s1-s1, 8
strncpy-s1-s1-mem, 9
strncpy-s1-s1-rfile, 8
strncpy-s1-sn, 9
strncpy-s1-sn-base, 8
strncpy-s1-sn-mem, 9
strncpy-s1-sn-rfile, 9
strncpy-s1-sn-rfile-base, 8
strncpy-s1p, 5, 7–9
strncpy-statep, 4, 6, 7, 12
strncpy-t, 3, 12
strncpy-t0, 3, 9
strncpy-t1, 3, 9, 10
strncpy-t2, 3, 12
strncpy1, 12

sub, 4–7, 13

uint-rangep, 5
uread-mem, 4

zero-list, 10
zero-list1, 9