```
**********************************************************
    9400 Thu May 29 16:56:01 2014
new/usr/src/man/man3c/semaphore.3c
4829 sema_init(3C) botches the arguments in an example
**********************************************************
   1 '\" te
   2 .\" Copyright (c) 2008 Sun Microsystems, Inc.  All Rights Reserved.
   3 .\" Portions Copyright (c) 2001, the Institute of Electrical and Electronics Eng
   4 .\" Portions Copyright (c) 1995 IEEE. All Rights Reserved
   5 .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
   6 .\" http://www.opengroup.org/bookstore/.
   7 .\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
   8 .\"  This notice shall appear on any product containing this material.
   9 .\" The contents of this file are subject to the terms of the Common Development
  10 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
  11 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
  12 .TH SEMAPHORE 3C "Feb 5, 2008"
  13 .SH NAME
  14 semaphore, sema_init, sema_destroy, sema_wait, sema_trywait, sema_post \-
  15 semaphores
  16 .SH SYNOPSIS
  17 .LP
  18 .nf
  19 cc [ \fIflag\fR... ] \fIfile\fR... -lthread  -lc [ \fIlibrary\fR... ]
  20 #include <synch.h>

  22 \fBint\fR \fBsema_init\fR(\fBsema_t *\fR\fIsp\fR, \fBunsigned int\fR \fIcount\fR
  23      \fBvoid *\fR \fIarg\fR);
  24 .fi

  26 .LP
  27 .nf
  28 \fBint\fR \fBsema_destroy\fR(\fBsema_t *\fR\fIsp\fR);
  29 .fi

  31 .LP
  32 .nf
  33 \fBint\fR \fBsema_wait\fR(\fBsema_t *\fR\fIsp\fR);
  34 .fi

  36 .LP
  37 .nf
  38 \fBint\fR \fBsema_trywait\fR(\fBsema_t *\fR\fIsp\fR);
  39 .fi

  41 .LP
  42 .nf
  43 \fBint\fR \fBsema_post\fR(\fBsema_t *\fR\fIsp\fR);
  44 .fi

  46 .SH DESCRIPTION
  47 .sp
  48 .LP
  49 A semaphore is a non-negative integer count and is generally used to coordinate
  50 access to resources. The initial semaphore count is set to the number of free
  51 resources, then threads slowly increment and decrement the count as resources
  52 are added and removed. If the semaphore count drops to 0, which means no
  53 available resources,  threads attempting to decrement the semaphore will block
  54 until the count is greater than 0.
  55 .sp
  56 .LP
  57 Semaphores can synchronize threads in this process and other processes if they
  58 are allocated in writable memory  and shared among the cooperating processes
  59 (see \fBmmap\fR(2)), and have been initialized for this purpose.
  60 .sp
  61 .LP
```



```
  62 Semaphores must be initialized before use; semaphores pointed to by \fIsp\fR to
  63 \fIcount\fR are initialized by \fBsema_init()\fR. The \fItype\fR argument can
  64 assign several different types of behavior to a semaphore. No current type uses
  65 \fIarg\fR, although it may be used in the future.
  66 .sp
  67 .LP
  68 The \fItype\fR argument may be one of the following:
  69 .sp
  70 .ne 2
  71 .na
  72 \fB\fBUSYNC_PROCESS\fR \fR
  73 .ad
  74 .RS 18n
  75 The semaphore can synchronize threads in this process and other processes.
  76 Initializing the semaphore should be done by only one process. A semaphore
  77 initialized with this type must be allocated in memory shared between
  78 processes, either in Sys V shared memory (see \fBshmop\fR(2)), or in memory
  79 mapped to a file (see \fBmmap\fR(2)). It is illegal to initialize the object
  80 this way and not allocate it in such shared memory. \fIarg\fR is ignored.
  81 .RE

  83 .sp
  84 .ne 2
  85 .na
  86 \fB\fBUSYNC_THREAD\fR \fR
  87 .ad
  88 .RS 18n
  89 The semaphore can synchronize  threads only in this process. The \fIarg\fR
  90 argument is ignored. \fBUSYNC_THREAD\fR does not support multiple mappings to
  91 the same logical synch object. If you need to \fBmmap()\fR a synch object to
  92 different locations within the same address space, then the synch object should
  93 be initialized as a shared object \fBUSYNC_PROCESS\fR for Solaris threads and
  94 \fBPTHREAD_PROCESS_PRIVATE\fR for POSIX threads.
  95 .RE

  97 .sp
  98 .LP
  99 A semaphore must not be simultaneously initialized by multiple threads, nor
 100 re-initialized while in use by other threads.
 101 .sp
 102 .LP
 103 Default semaphore initialization (intra-process):
 104 .sp
 105 .in +2
 106 .nf
 107 sema_t sp;
 108 int count  =  1;
 109 sema_init(&sp, count, 0, NULL);
 109 sema_init(&sp, count, NULL, NULL);
 110 .fi
 111 .in -2

 113 .sp
 114 .LP
 115 or
 116 .sp
 117 .in +2
 118 .nf
 119 sema_init(&sp, count, USYNC_THREAD, NULL);
 120 .fi
 121 .in -2

 123 .sp
 124 .LP
 125 Customized semaphore initialization (inter-process):
 126 .sp
```

```
.in +2
.nf
\fBsema_t sp;
int count  =  1;
sema_init(&sp, count, USYNC_PROCESS, NULL);\fR
.fi
.in -2

.sp
.LP
The \fBsema_destroy()\fR function destroys any state related to the semaphore
pointed to by \fIsp\fR. The semaphore storage space is not released.
.sp
.LP
The \fBsema_wait()\fR function blocks the calling thread until the semaphore
count pointed to by \fIsp\fR is greater than 0, and then it atomically
decrements the count.
.sp
.LP
The \fBsema_trywait()\fR function atomically decrements the semaphore count
pointed to by \fIsp\fR, if the count is greater than 0; otherwise, it returns
an error.
.sp
.LP
The \fBsema_post()\fR function atomically increments the semaphore count
pointed to by \fIsp\fR. If there are any threads blocked on the semaphore, one
will be unblocked.
.sp
.LP
The semaphore functionality described on this man page is for the Solaris
threads implementation.  For the POSIX-conforming semaphore interface
documentation, see \fBsem_close\fR(3C), \fBsem_destroy\fR(3C),
\fBsem_getvalue\fR(3C), \fBsem_init\fR(3C), \fBsem_open\fR(3C),
\fBsem_post\fR(3C), \fBsem_unlink\fR(3C), and \fBsem_wait\fR(3C).
.SH RETURN VALUES
.sp
.LP
Upon successful completion,  \fB0\fR is returned; otherwise, a non-zero value
indicates an error.
.SH ERRORS
.sp
.LP
These functions will fail if:
.sp
.ne 2
.na
\fB\fBEINVAL\fR \fR
.ad
.RS 11n
The \fIsp\fR argument does not refer to a valid semaphore.
.RE

.sp
.ne 2
.na
\fB\fBEFAULT\fR \fR
.ad
.RS 11n
Either the \fIsp\fR or \fIarg\fR argument points to an illegal address.
.RE

.sp
.LP
The \fBsema_wait()\fR function will fail if:
.sp
.ne 2
```



```
.na
\fB\fBEINTR\fR \fR
.ad
.RS 10n
The wait was interrupted by a signal or \fBfork()\fR.
.RE

.sp
.LP
The \fBsema_trywait()\fR function will fail if:
.sp
.ne 2
.na
\fB\fBEBUSY\fR \fR
.ad
.RS 10n
The semaphore pointed to by \fIsp\fR has a 0 count.
.RE

.sp
.LP
The \fBsema_post()\fR function will fail if:
.sp
.ne 2
.na
\fB\fBEOVERFLOW\fR \fR
.ad
.RS 14n
The semaphore value pointed to by \fIsp\fR exceeds \fBSEM_VALUE_MAX\fR.
.RE

.SH EXAMPLES
.LP
\fBExample 1 \fRThe customer waiting-line in a bank is analogous to the
synchronization scheme of a semaphore using \fBsema_wait()\fR and
\fBsema_trywait()\fR:
.sp
.in +2
.nf
/* cc [ flag \|.\|.\|. ] file \|.\|.\|. -lthread [ library \|.\|.\|. ] */
#include <errno.h>
#define TELLERS 10
sema_t     tellers;     /* semaphore */
int banking_hours(), deposit_withdrawal;
void*customer(), do_business(), skip_banking_today();
\&.\|.\|.

sema_init(&tellers, TELLERS, USYNC_THREAD, NULL);
    /* 10 tellers available */
while(banking_hours())
    pthread_create(NULL, NULL, customer, deposit_withdrawal);
\&.\|.\|.

void *
customer(int deposit_withdrawal)
{
     int this_customer, in_a_hurry = 50;
     this_customer = rand() % 100;

     if (this_customer == in_a_hurry)  {
         if (sema_trywait(&tellers) != 0)
             if (errno == EBUSY){ /* no teller available */
                  skip_banking_today(this_customer);
                  return;
         } /* else go immediately to available teller and
                                  decrement tellers */
```

```
 259       }
 260       else
 261          sema_wait(&tellers); /* wait for next teller, then
 262                                  proceed, and decrement tellers */

 264       do_business(deposit_withdrawal);
 265       sema_post(&tellers); /* increment tellers; this_customer's
 266                               teller is now available */
 267 }
```

_____***unchanged_portion_omitted_***