Static Driver Verifier – Introduction

Byron Cook
Software Design Engineer
Windows Device Experience Group
Bycook @ microsoft.com
Microsoft Corporation

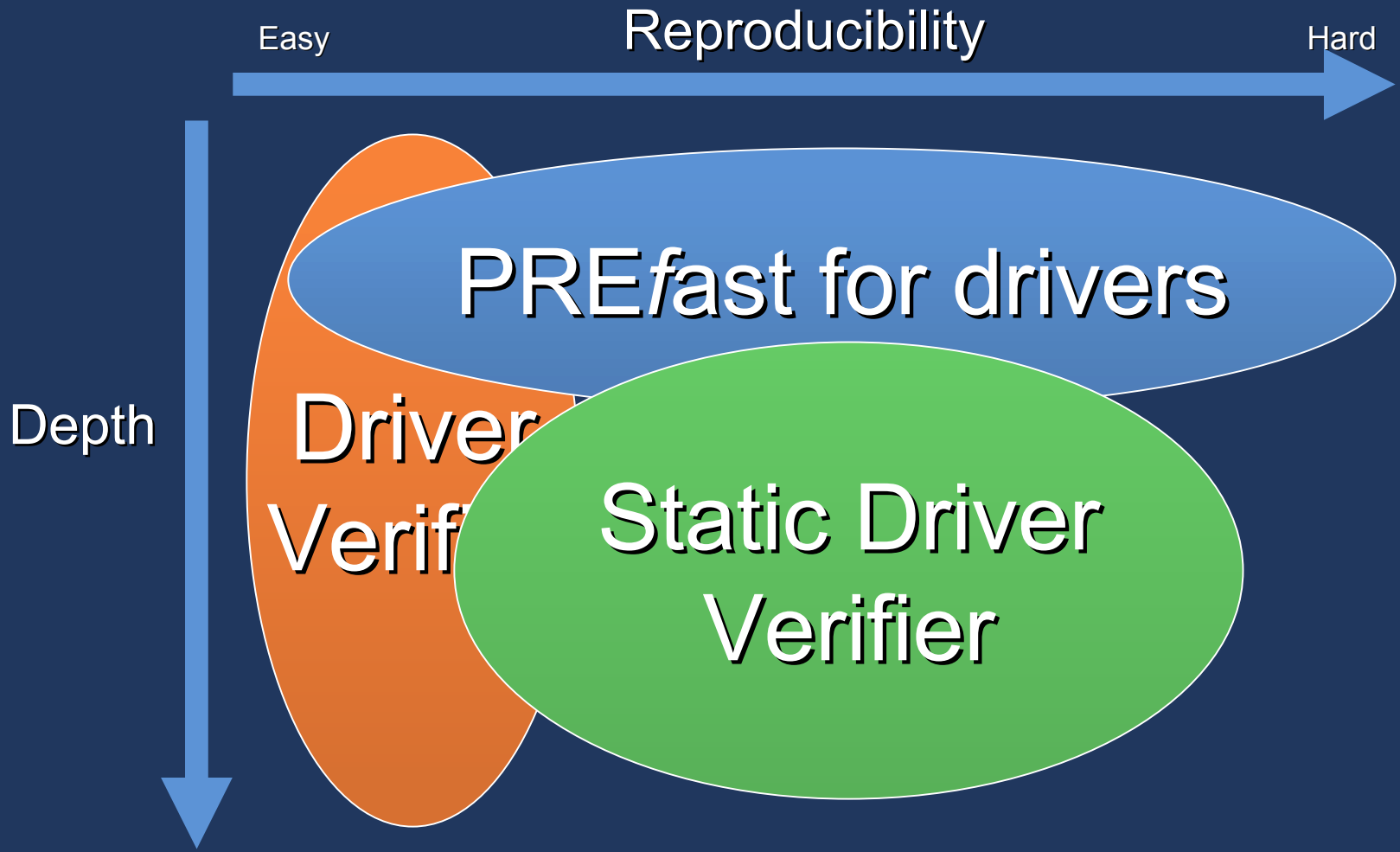
Session Outline

- Introduction to Static Driver Verifier
- Static Driver Verifier internals
- Summary
- Demo
- Question and Answer

Introduction To Static Driver Verifier

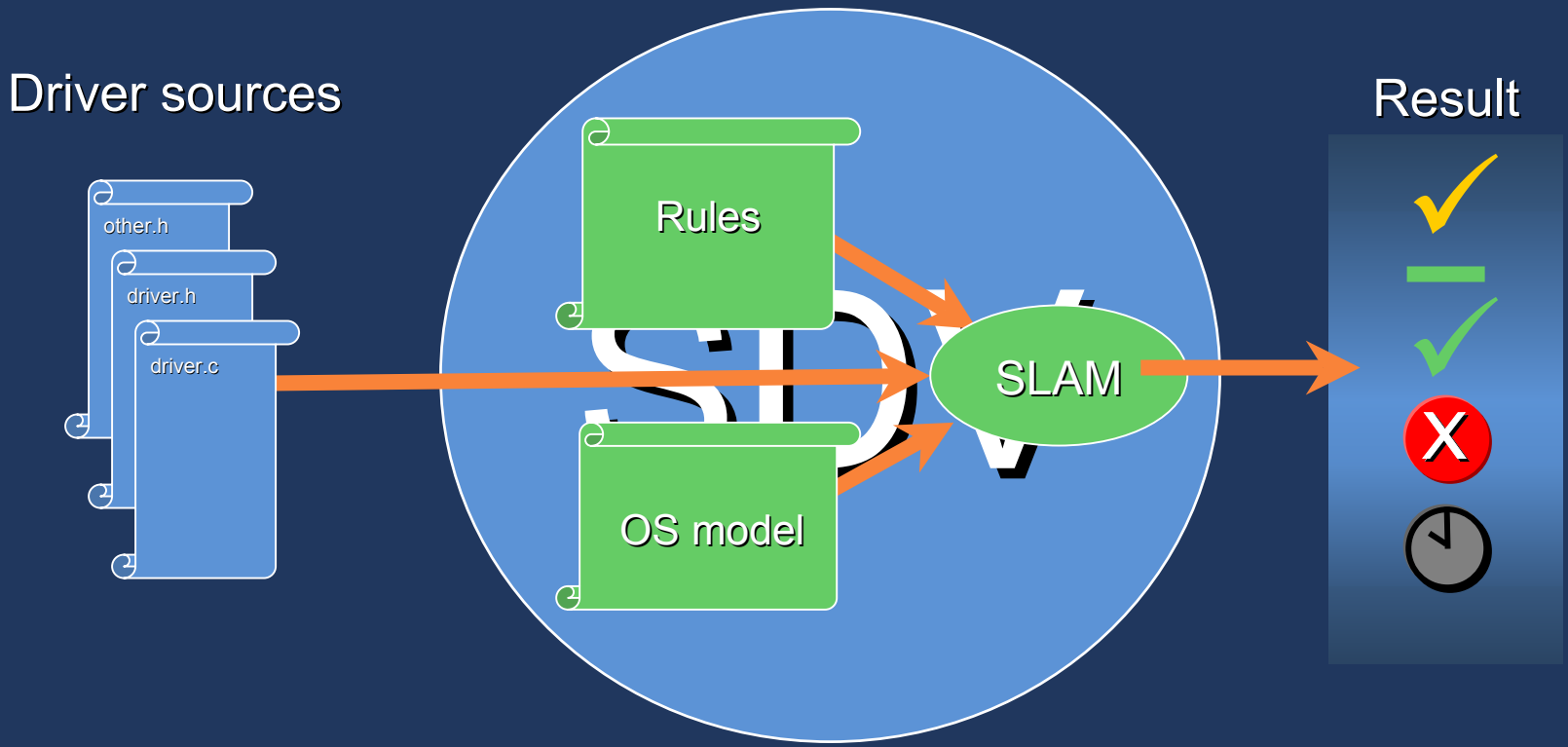
- Static Driver Verifier (SDV) is a tool for finding bugs in drivers
- SDV is (currently) applicable only to WDM drivers
- SDV operates on the driver's source code
- SDV is completely automatic
- SDV checks that drivers do not violate a set of "kernel API usage rules"

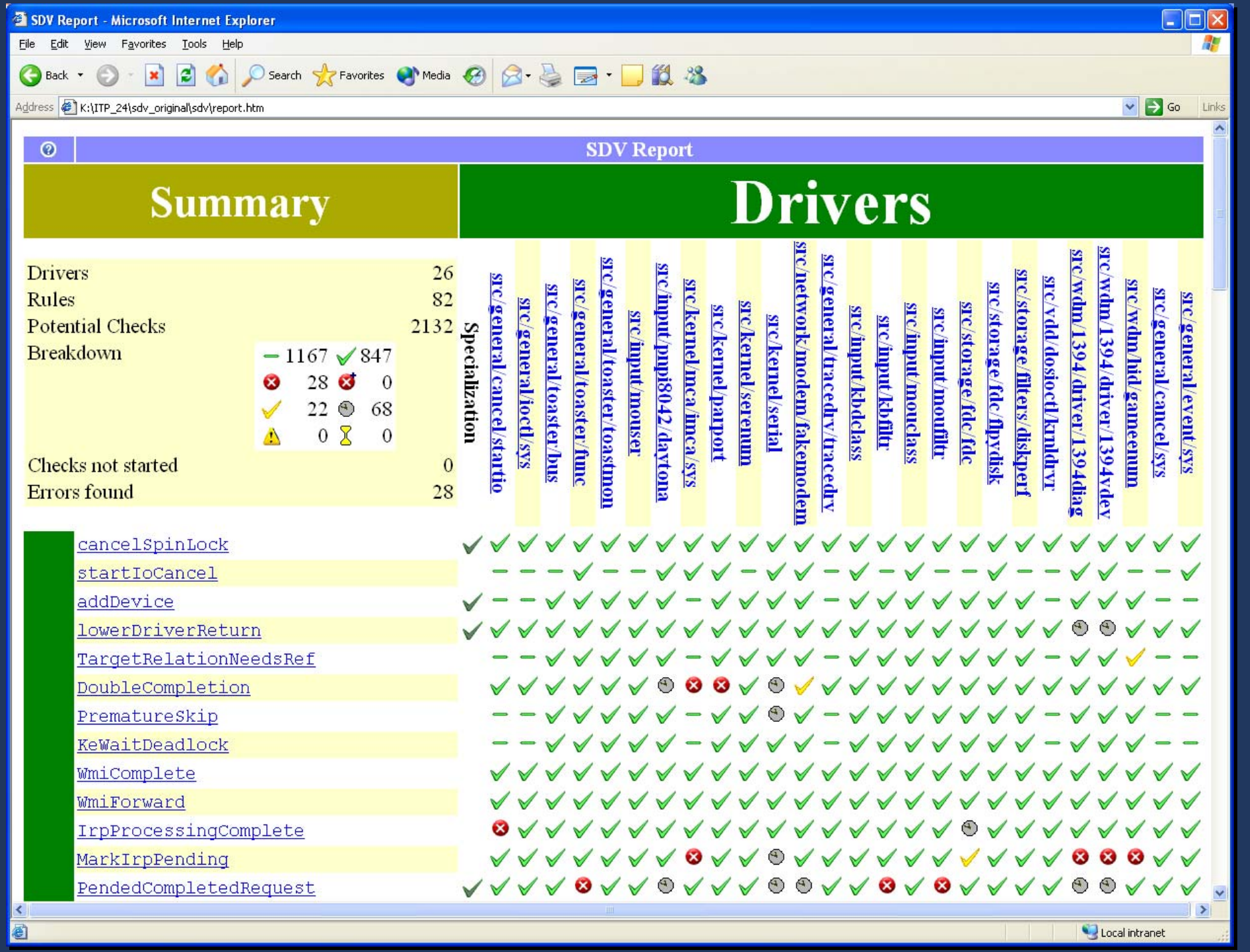
Introduction To Static Driver Verifier

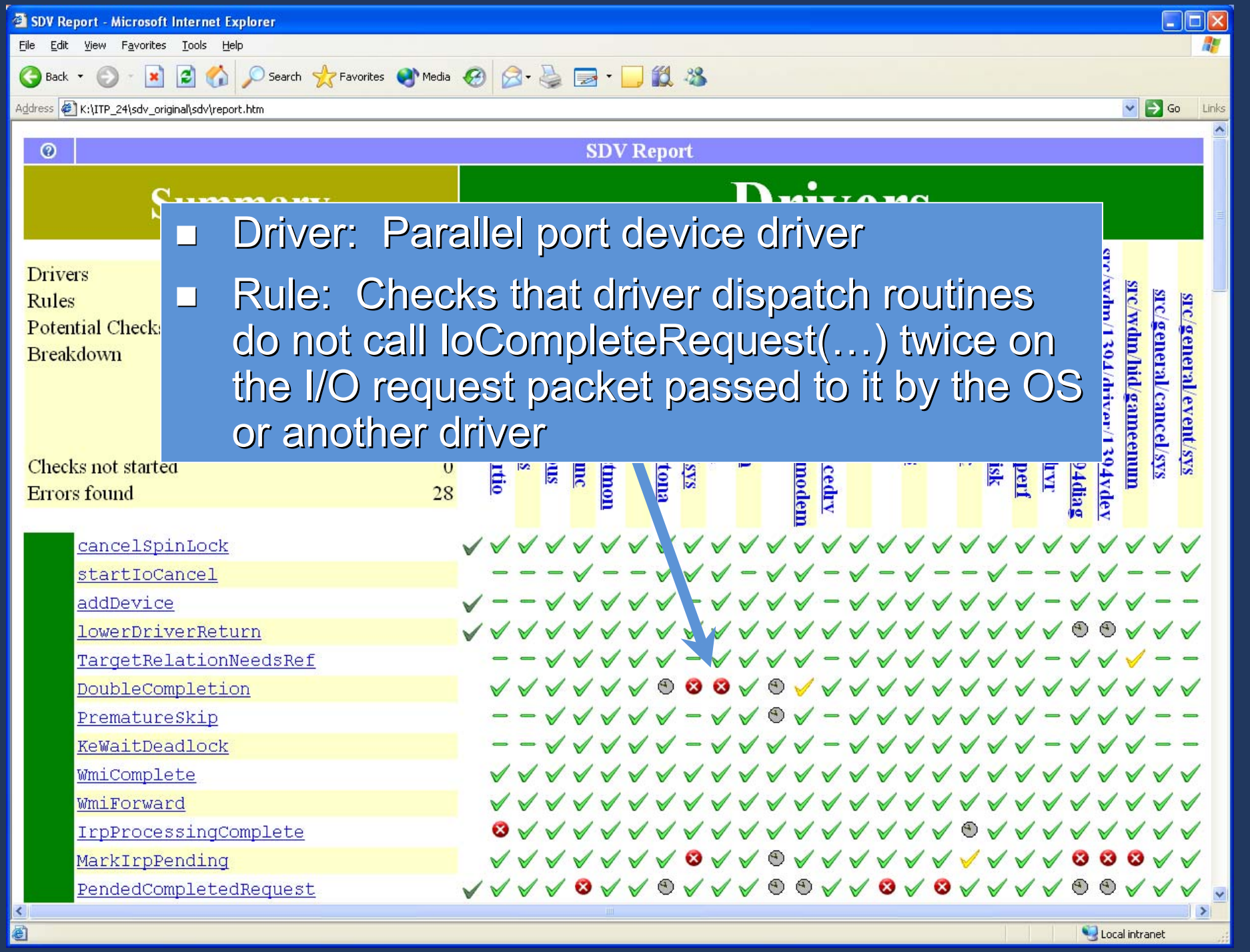


Introduction To Static Driver Verifier

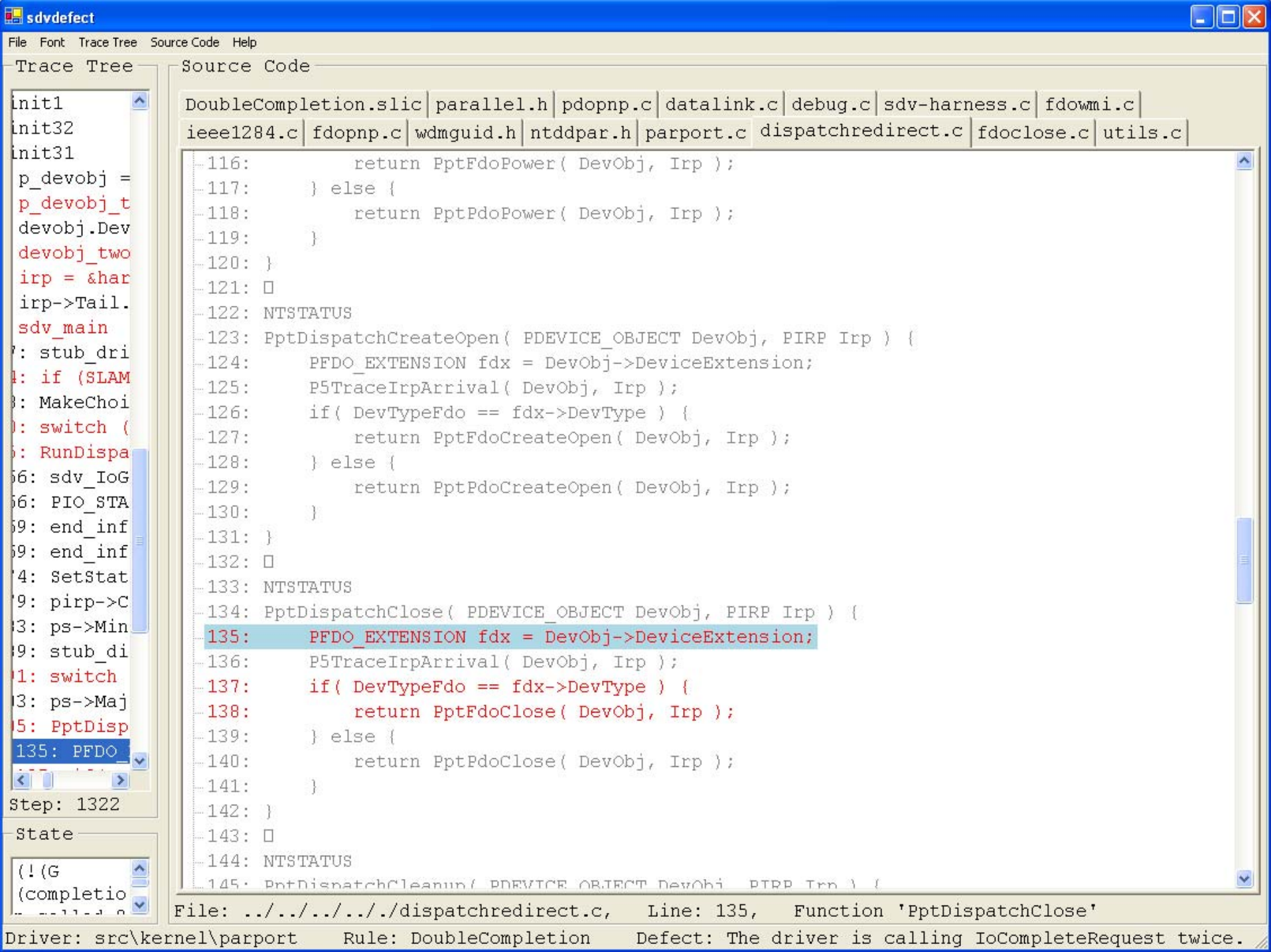
Introduction To Static Driver Verifier

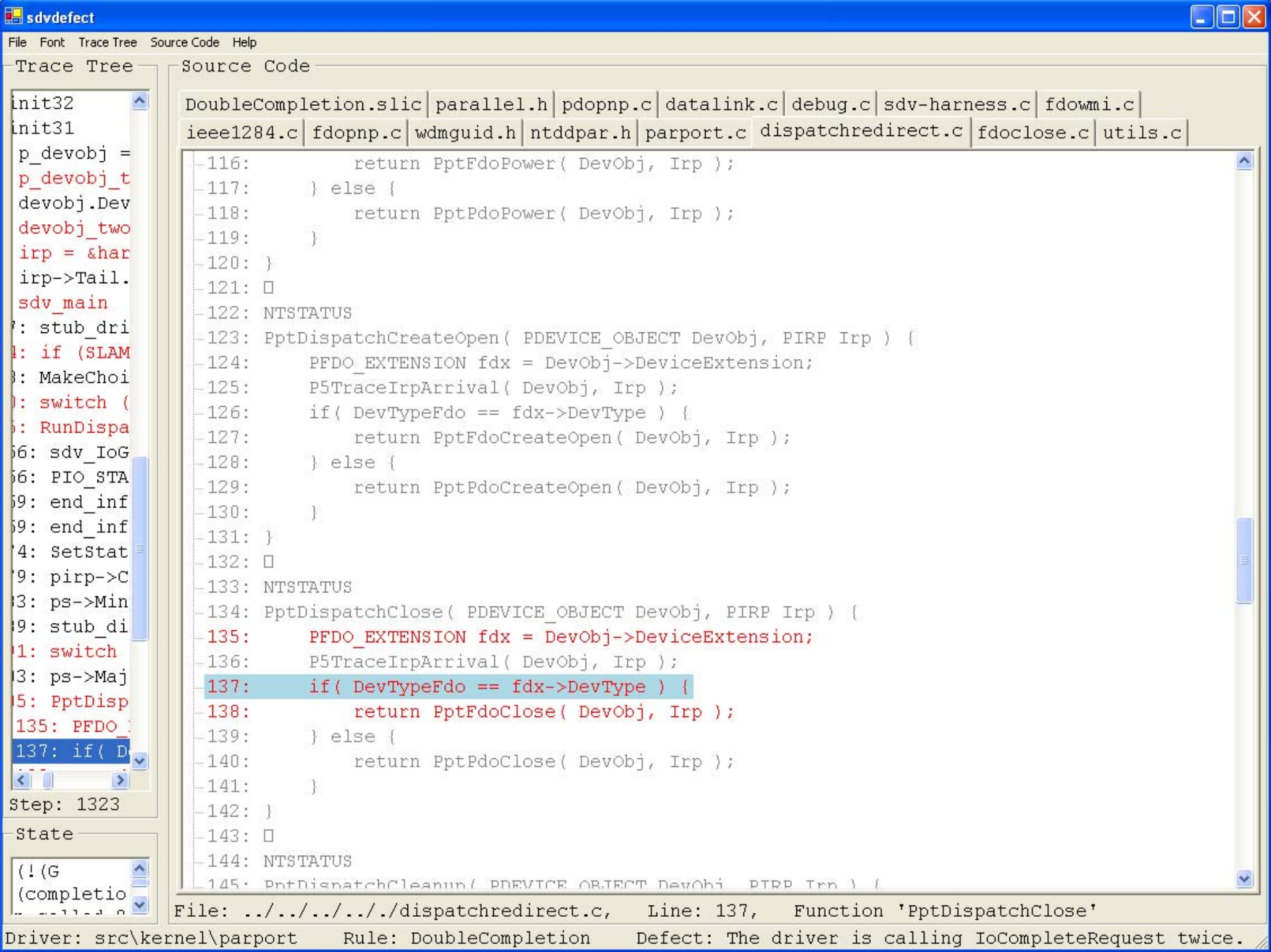


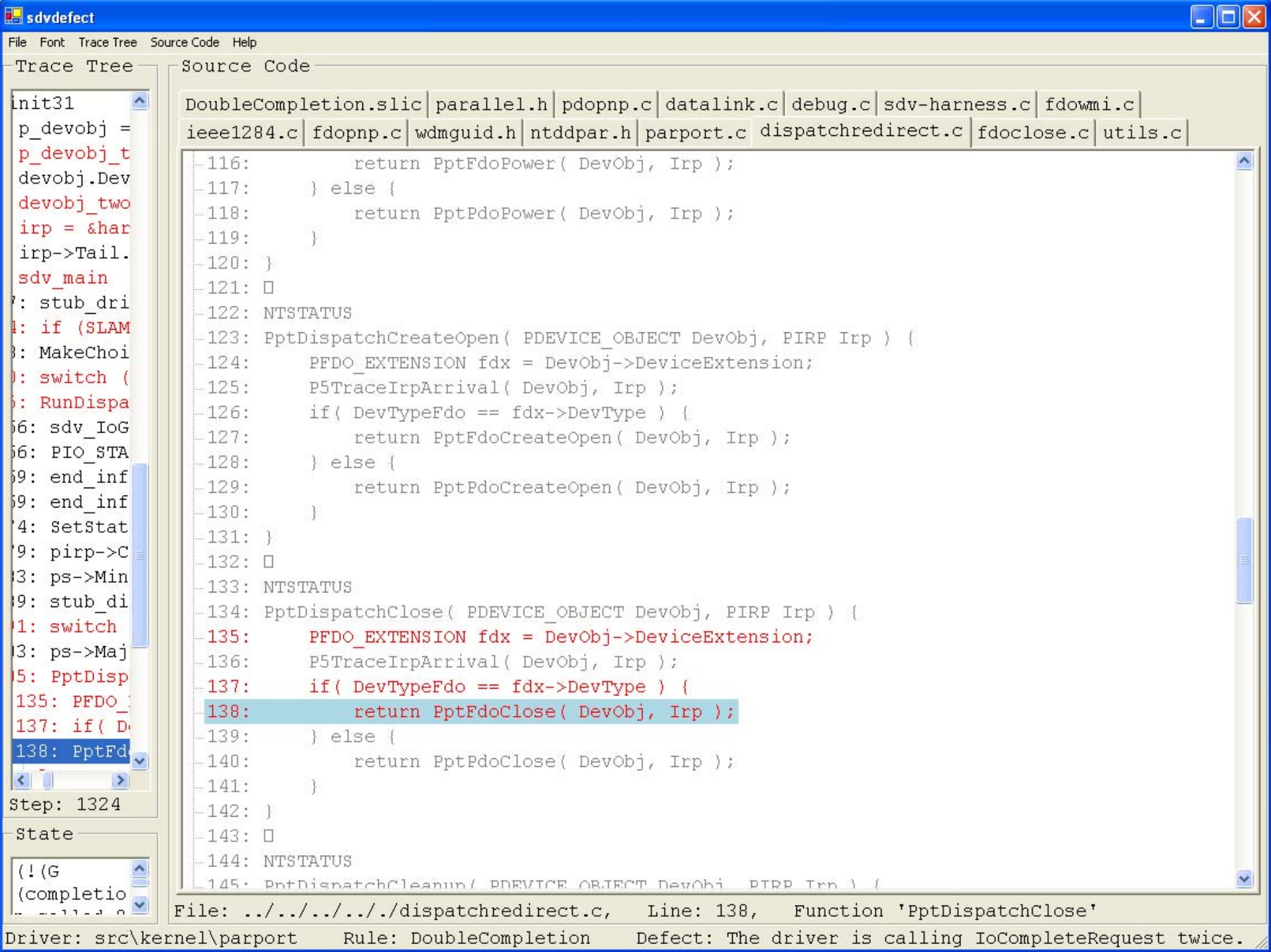

SDV Report - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address K:\ITP_24\sdv_original\sdv\report.htm
Go Links
SDV Report
Summary
Drivers
Drivers 26
Rules 82
Potential Checks 2132
Breakdown
1167 847
28 0
22 68
0 0
Checks not started 0
Errors found 28
Specialization
src/general/cancel/startio
src/general/ioctl/sys
src/general/toaster/bus
src/general/toaster/func
src/general/toaster/toastmon
src/input/mouser
src/input/pnpi8042/daytona
src/kernel/mca/imca/sys
src/kernel/parport
src/kernel/serenum
src/kernel/serial
src/network/modem/fakemodem
src/general/tracedrv/tracedrv
src/input/kbdclass
src/input/kbfiltr
src/input/mouclass
src/input/moufiltr
src/storage/fdc/fdc
src/storage/fdc/flpydisk
src/storage/filters/diskperf
src/vdd/dosioctl/krnldrvr
src/wdm/1394/driver/1394diag
src/wdm/1394/driver/1394vdev
src/wdm/hid/gameenum
src/general/cancel/sys
src/general/event/sys
cancelSpinLock
startIoCancel
addDevice
lowerDriverReturn
TargetRelationNeedsRef
DoubleCompletion
PrematureSkip
KeWaitDeadlock
WmiComplete
WmiForward
IrpProcessingComplete
MarkIrpPending
PendedCompletedRequest
Local intranet



- Driver: Parallel port device driver
- Rule: Checks that driver dispatch routines do not call IoCompleteRequest(…) twice on the I/O request packet passed to it by the OS or another driver


sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
init1
init32
init31
p_devobj =
p_devobj_t
devobj.Dev
devobj_two
irp = &har
irp->Tail.
sdv_main
7: stub_dri
4: if (SLAM
3: MakeChoi
): switch (
5: RunDispa
56: sdv_IoG
56: PIO_STA
59: end_inf
59: end_inf
74: SetStat
79: pirp->C
83: ps->Min
89: stub_di
91: switch
03: ps->Maj
05: PptDisp
135: PFDO_
Step: 1322
State
(!(G
(completio
Source Code
DoubleCompletion.slic
parallel.h
pdopnp.c
datalink.c
debug.c
sdv-harness.c
fdowmi.c
ieee1284.c
fdopnp.c
wdmguid.h
ntddpar.h
parport.c
dispatchredirect.c
fdoclose.c
utils.c
116: return PptFdoPower(DevObj, Irp);
117: } else {
118: return PptPdoPower(DevObj, Irp);
119: }
120: }
121:
122: NTSTATUS
123: PptDispatchCreateOpen(PDEVICE_OBJECT DevObj, PIRP Irp) {
124: PFDO_EXTENSION fdx = DevObj->DeviceExtension;
125: P5TraceIrpArrival(DevObj, Irp);
126: if(DevTypeFdo == fdx->DevType) {
127: return PptFdoCreateOpen(DevObj, Irp);
128: } else {
129: return PptPdoCreateOpen(DevObj, Irp);
130: }
131: }
132:
133: NTSTATUS
134: PptDispatchClose(PDEVICE_OBJECT DevObj, PIRP Irp) {
135: PFDO_EXTENSION fdx = DevObj->DeviceExtension;
136: P5TraceIrpArrival(DevObj, Irp);
137: if(DevTypeFdo == fdx->DevType) {
138: return PptFdoClose(DevObj, Irp);
139: } else {
140: return PptPdoClose(DevObj, Irp);
141: }
142: }
143:
144: NTSTATUS
File: ../../../.././dispatchredirect.c, Line: 135, Function 'PptDispatchClose'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.


sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
init32
init31
p_devobj =
p_devobj_t
devobj.Dev
devobj_two
irp = &har
irp->Tail.
sdv_main
7: stub_dri
4: if (SLAM
3: MakeChoi
0: switch (
5: RunDispa
56: sdv_IoG
56: PIO_STA
59: end_inf
59: end_inf
74: SetStat
79: pirp->C
83: ps->Min
89: stub_di
91: switch
93: ps->Maj
95: PptDisp
135: PFDO_
137: if(D
Step: 1323
State
(!(G
(completio
Source Code
DoubleCompletion.slic
parallel.h
pdopnp.c
datalink.c
debug.c
sdv-harness.c
fdowmi.c
ieee1284.c
fdopnp.c
wdmguid.h
ntddpar.h
parport.c
dispatchredirect.c
fdoclose.c
utils.c
116: return PptFdoPower(DevObj, Irp);
117: } else {
118: return PptPdoPower(DevObj, Irp);
119: }
120: }
121:
122: NTSTATUS
123: PptDispatchCreateOpen(PDEVICE_OBJECT DevObj, PIRP Irp) {
124: PFDO_EXTENSION fdx = DevObj->DeviceExtension;
125: P5TraceIrpArrival(DevObj, Irp);
126: if(DevTypeFdo == fdx->DevType) {
127: return PptFdoCreateOpen(DevObj, Irp);
128: } else {
129: return PptPdoCreateOpen(DevObj, Irp);
130: }
131: }
132:
133: NTSTATUS
134: PptDispatchClose(PDEVICE_OBJECT DevObj, PIRP Irp) {
135: PFDO_EXTENSION fdx = DevObj->DeviceExtension;
136: P5TraceIrpArrival(DevObj, Irp);
137: if(DevTypeFdo == fdx->DevType) {
138: return PptFdoClose(DevObj, Irp);
139: } else {
140: return PptPdoClose(DevObj, Irp);
141: }
142: }
143:
144: NTSTATUS
File: ../../../.././dispatchredirect.c, Line: 137, Function 'PptDispatchClose'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.


sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
init31
p_devobj =
p_devobj_t
devobj.Dev
devobj_two
irp = &har
irp->Tail.
sdv_main
7: stub_dri
4: if (SLAM
3: MakeChoi
0: switch (
5: RunDispa
56: sdv_IoG
56: PIO_STA
59: end_inf
59: end_inf
74: SetStat
79: pirp->C
83: ps->Min
89: stub_di
91: switch
03: ps->Maj
05: PptDisp
135: PFDO_
137: if(D
138: PptFd
Step: 1324
State
(!(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
116: return PptFdoPower(DevObj, Irp);
117: } else {
118: return PptPdoPower(DevObj, Irp);
119: }
120: }
121:
122: NTSTATUS
123: PptDispatchCreateOpen(PDEVICE_OBJECT DevObj, PIRP Irp) {
124: PFDO_EXTENSION fdx = DevObj->DeviceExtension;
125: P5TraceIrpArrival(DevObj, Irp);
126: if(DevTypeFdo == fdx->DevType) {
127: return PptFdoCreateOpen(DevObj, Irp);
128: } else {
129: return PptPdoCreateOpen(DevObj, Irp);
130: }
131: }
132:
133: NTSTATUS
134: PptDispatchClose(PDEVICE_OBJECT DevObj, PIRP Irp) {
135: PFDO_EXTENSION fdx = DevObj->DeviceExtension;
136: P5TraceIrpArrival(DevObj, Irp);
137: if(DevTypeFdo == fdx->DevType) {
138: return PptFdoClose(DevObj, Irp);
139: } else {
140: return PptPdoClose(DevObj, Irp);
141: }
142: }
143:
144: NTSTATUS
File: ../../../.././dispatchredirect.c, Line: 138, Function 'PptDispatchClose'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

sdvdefect

File Font Trace Tree Source Code Help

Trace Tree

```
RunDispatc
 sdv_IoGet
 PIO_STACK
 end_info
 end_info
 SetStatus
 pirp->Can
 ps->Minor
 stub_disp
 switch (x
 ps->Major
 PptDispat
5: PFDO_EX
7: if( Dev
8: PptFdoC
9: PFDO_EX
12: do_pag
19: if( fd
24: P4Comp
  1782: Irp
  1783: Irp
+ 1784: SL
  1784: sd
  1785: re
  1785: Re
63: P4Comp
- 1797: P4
```

Step: 1327

State

```
(!(G
(completio
```

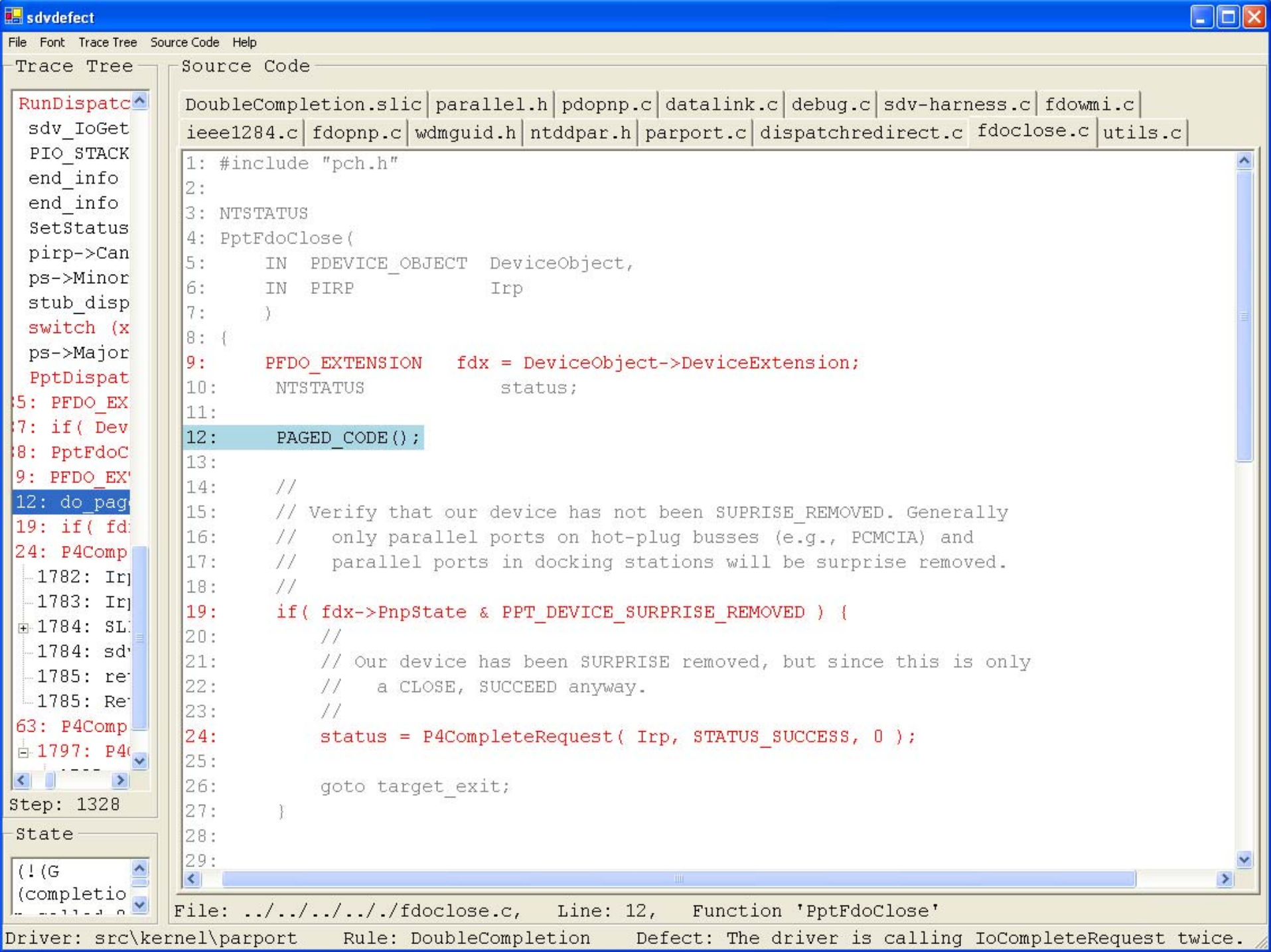
Source Code

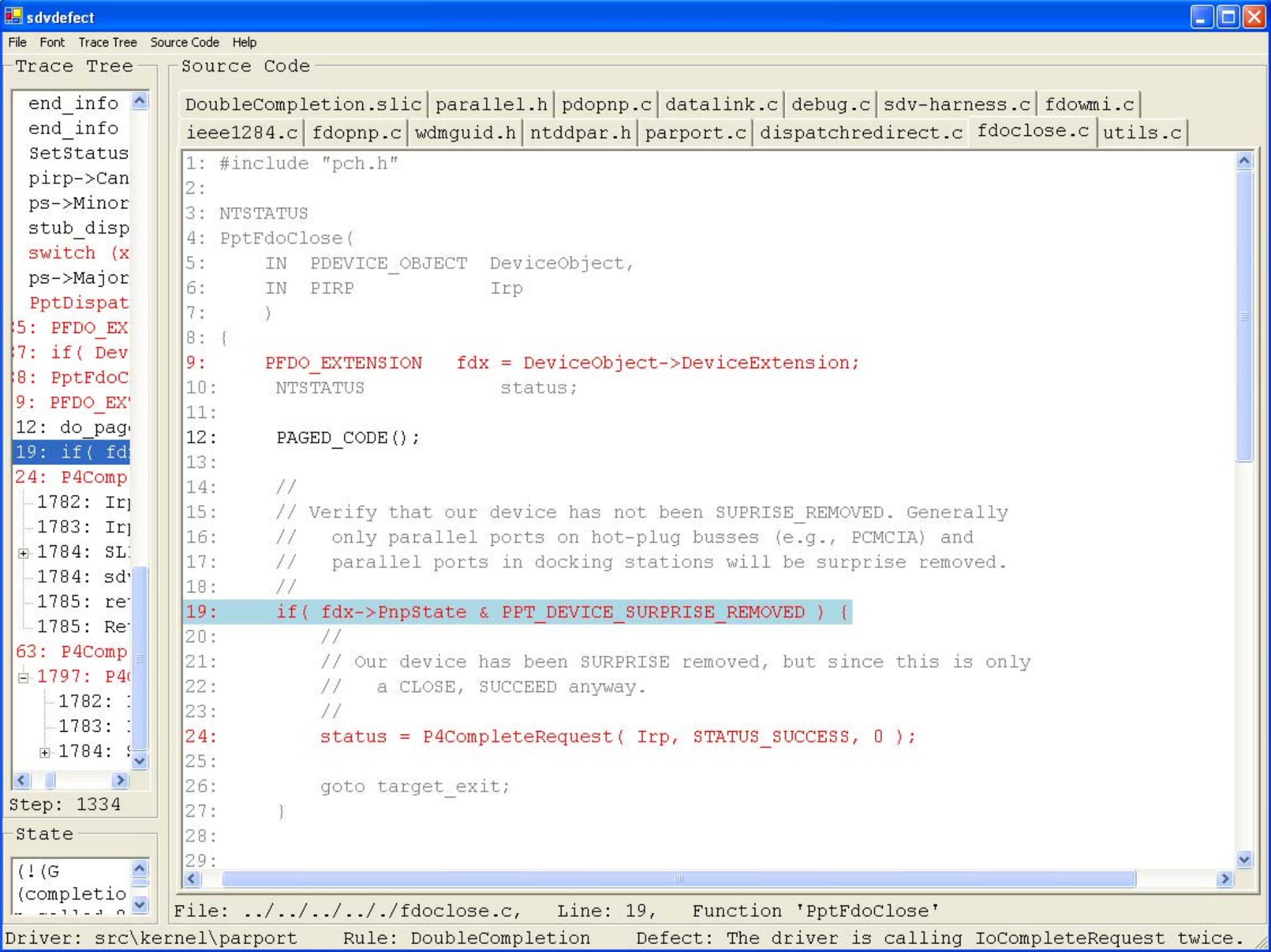
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c | ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c

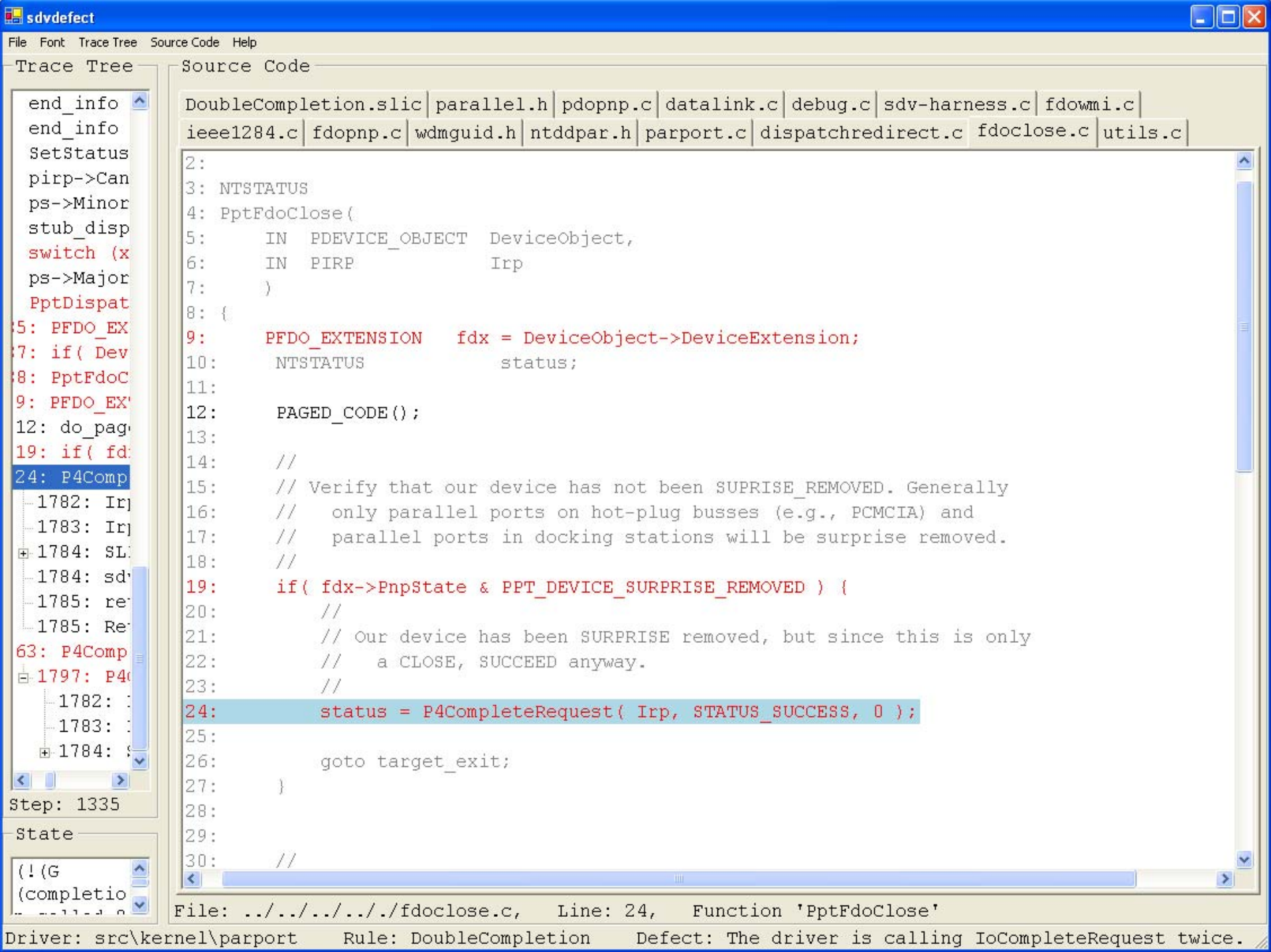
```
1:  #include "pch.h"
2:
3:  NTSTATUS
4:  PptFdoClose(
5:      IN  PDEVICE_OBJECT  DeviceObject,
6:      IN  PIRP            Irp
7:      )
8:  {
9:      PFDO_EXTENSION    fdx = DeviceObject->DeviceExtension;
10:     NTSTATUS            status;
11:
12:     PAGED_CODE();
13:
14:     //
15:     // Verify that our device has not been SUPRISE_REMOVED. Generally
16:     //   only parallel ports on hot-plug busses (e.g., PCMCIA) and
17:     //   parallel ports in docking stations will be surprise removed.
18:     //
19:     if( fdx->PnpState & PPT_DEVICE_SURPRISE_REMOVED ) {
20:         //
21:         // Our device has been SURPRISE removed, but since this is only
22:         //   a CLOSE, SUCCEED anyway.
23:         //
24:         status = P4CompleteRequest( Irp, STATUS_SUCCESS, 0 );
25:
26:         goto target_exit;
27:     }
28:
29:
```

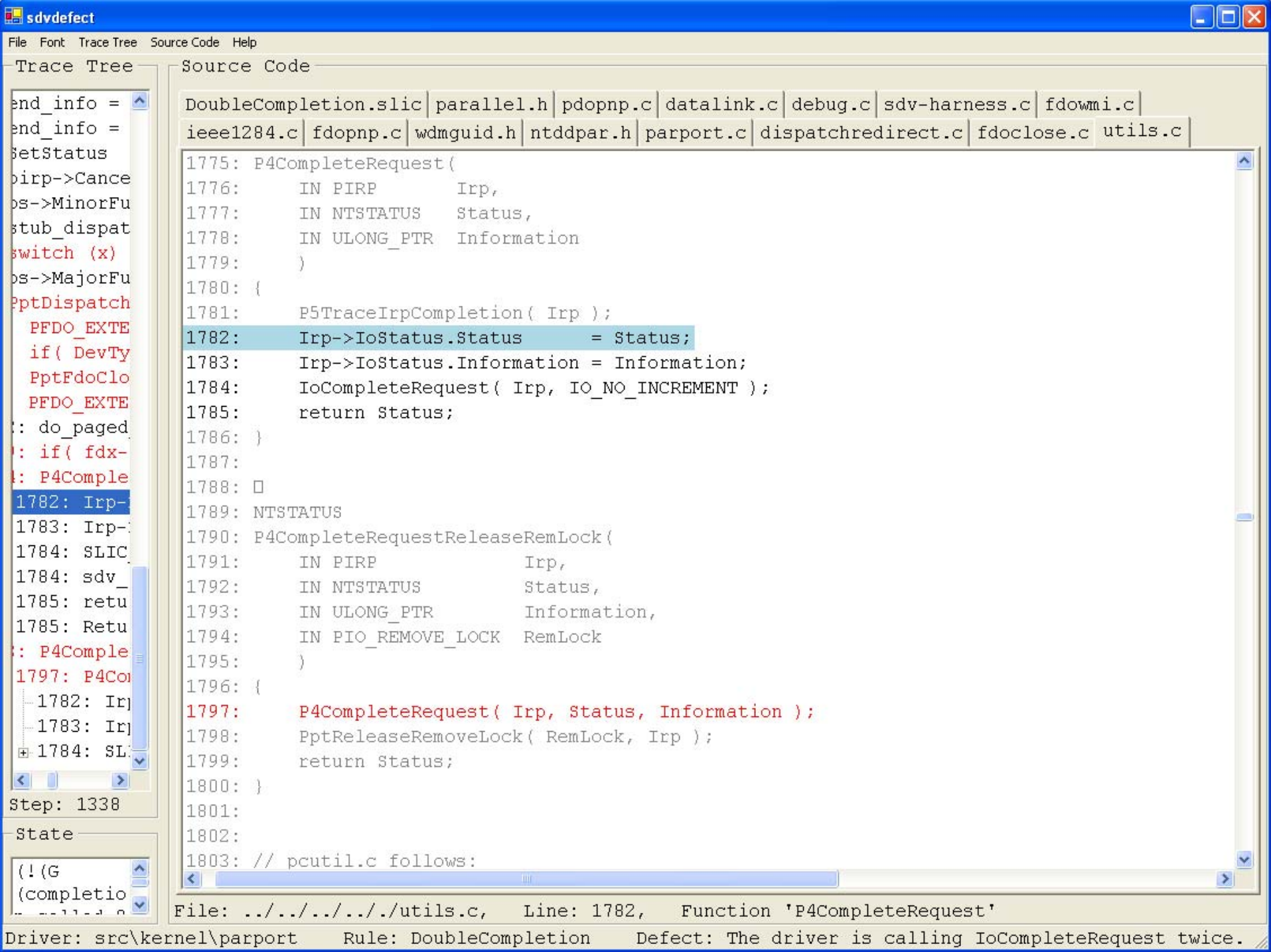
File: ../../../.././fdoclose.c, Line: 9, Function 'PptFdoClose'

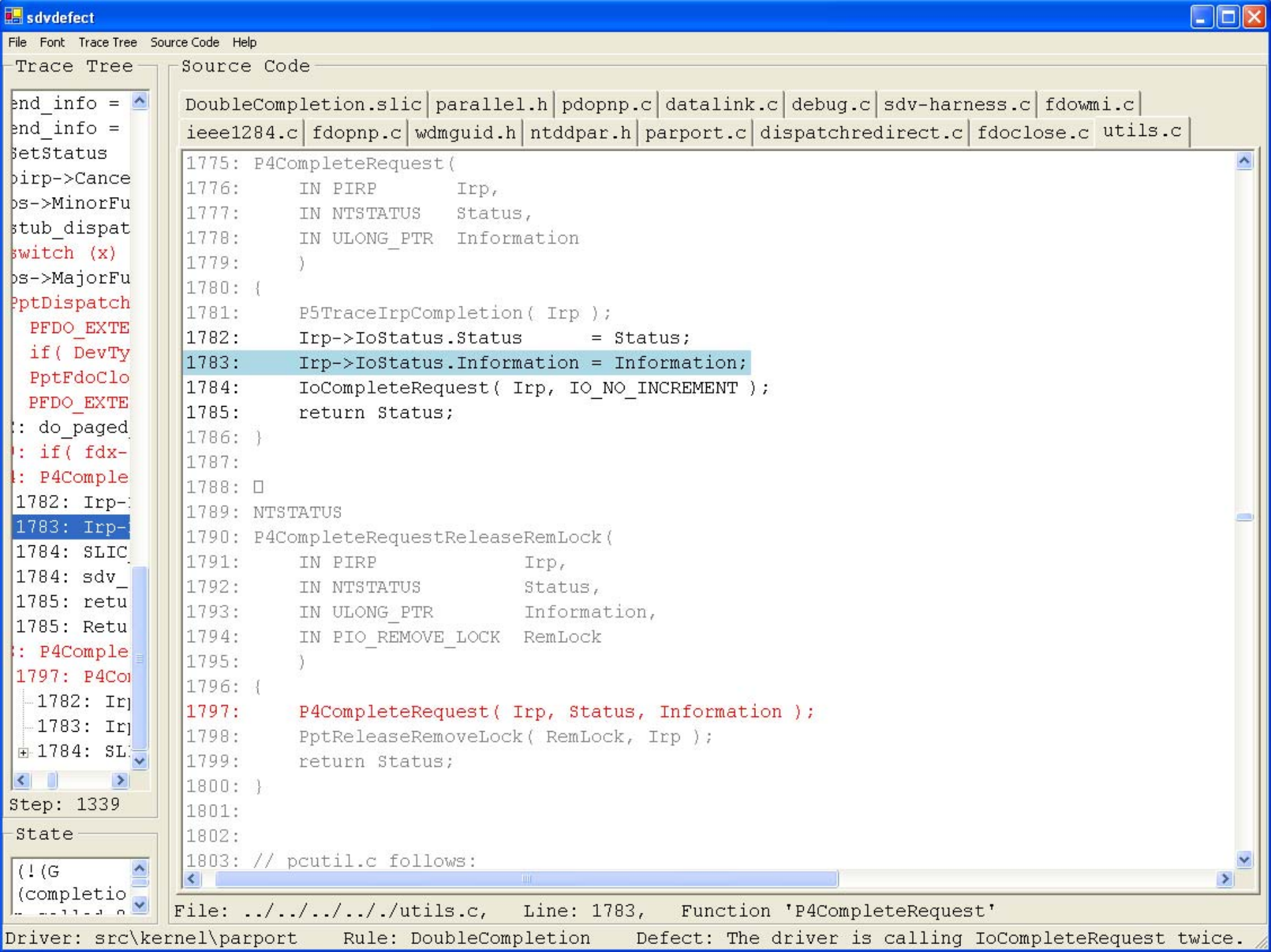
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

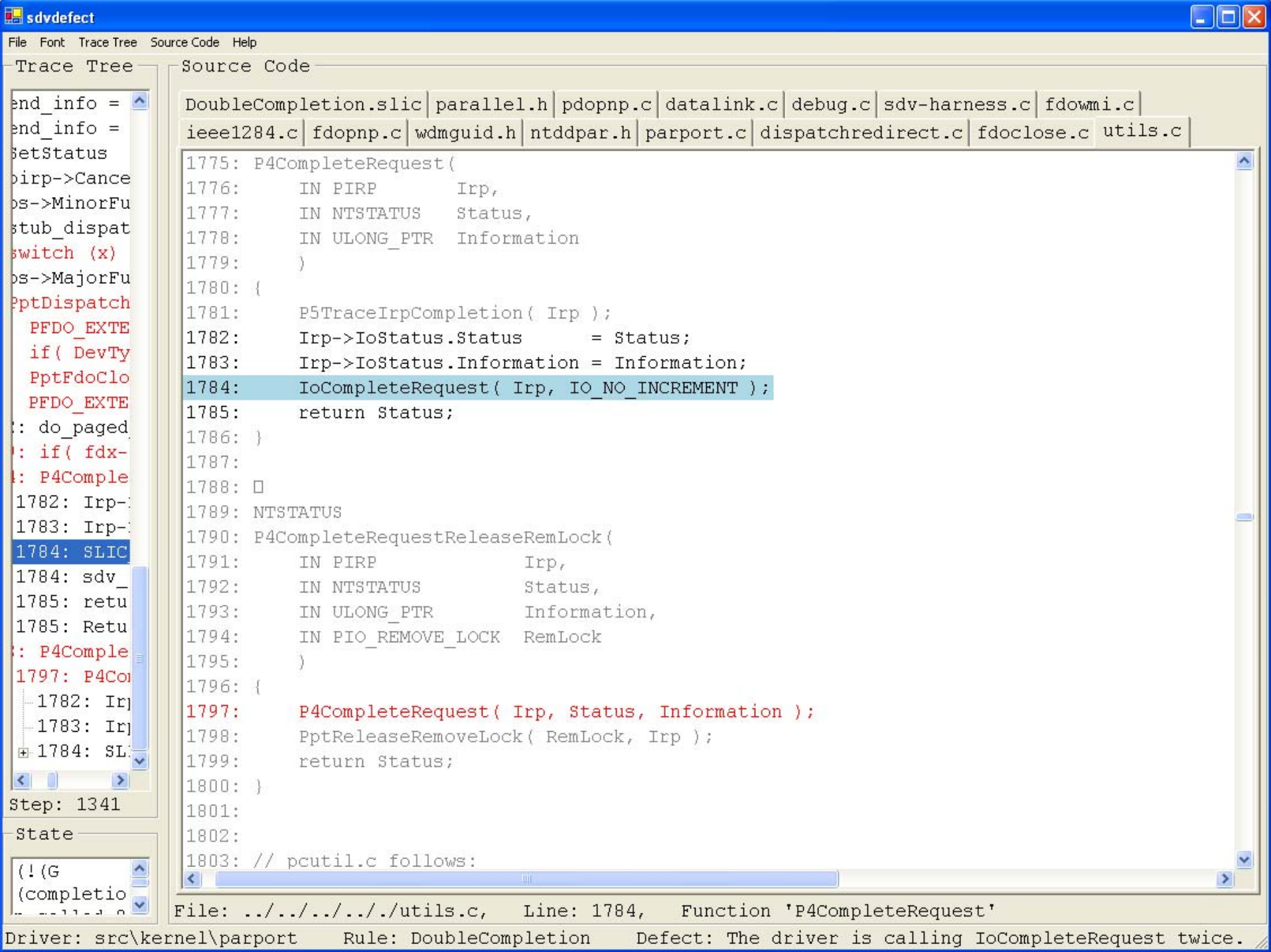

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
RunDispatc
sdv_IoGet
PIO_STACK
end_info
end_info
SetStatus
pirp->Can
ps->Minor
stub_disp
switch (x
ps->Major
PptDispat
5: PFDO_EX
7: if(Dev
8: PptFdoC
9: PFDO_EX
12: do_pag
19: if(fd
24: P4Comp
1782: Irj
1783: Irj
1784: SL
1784: sd
1785: re
1785: Re
63: P4Comp
1797: P4
Step: 1328
State
(!(G
(completio
Source Code
DoubleCompletion.slic
parallel.h
pdopnp.c
datalink.c
debug.c
sdv-harness.c
fdowmi.c
ieee1284.c
fdopnp.c
wdmguid.h
ntddpar.h
parport.c
dispatchredirect.c
fdoclose.c
utils.c
1: #include "pch.h"
2:
3: NTSTATUS
4: PptFdoClose(
5: IN PDEVICE_OBJECT DeviceObject,
6: IN PIRP Irp
7:)
8: {
9: PFDO_EXTENSION fdx = DeviceObject->DeviceExtension;
10: NTSTATUS status;
11:
12: PAGED_CODE();
13:
14: //
15: // Verify that our device has not been SUPRISE_REMOVED. Generally
16: // only parallel ports on hot-plug busses (e.g., PCMCIA) and
17: // parallel ports in docking stations will be surprise removed.
18: //
19: if(fdx->PnpState & PPT_DEVICE_SURPRISE_REMOVED) {
20: //
21: // Our device has been SURPRISE removed, but since this is only
22: // a CLOSE, SUCCEED anyway.
23: //
24: status = P4CompleteRequest(Irp, STATUS_SUCCESS, 0);
25:
26: goto target_exit;
27: }
28:
29:
File: ../../../.././fdoclose.c, Line: 12, Function 'PptFdoClose'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

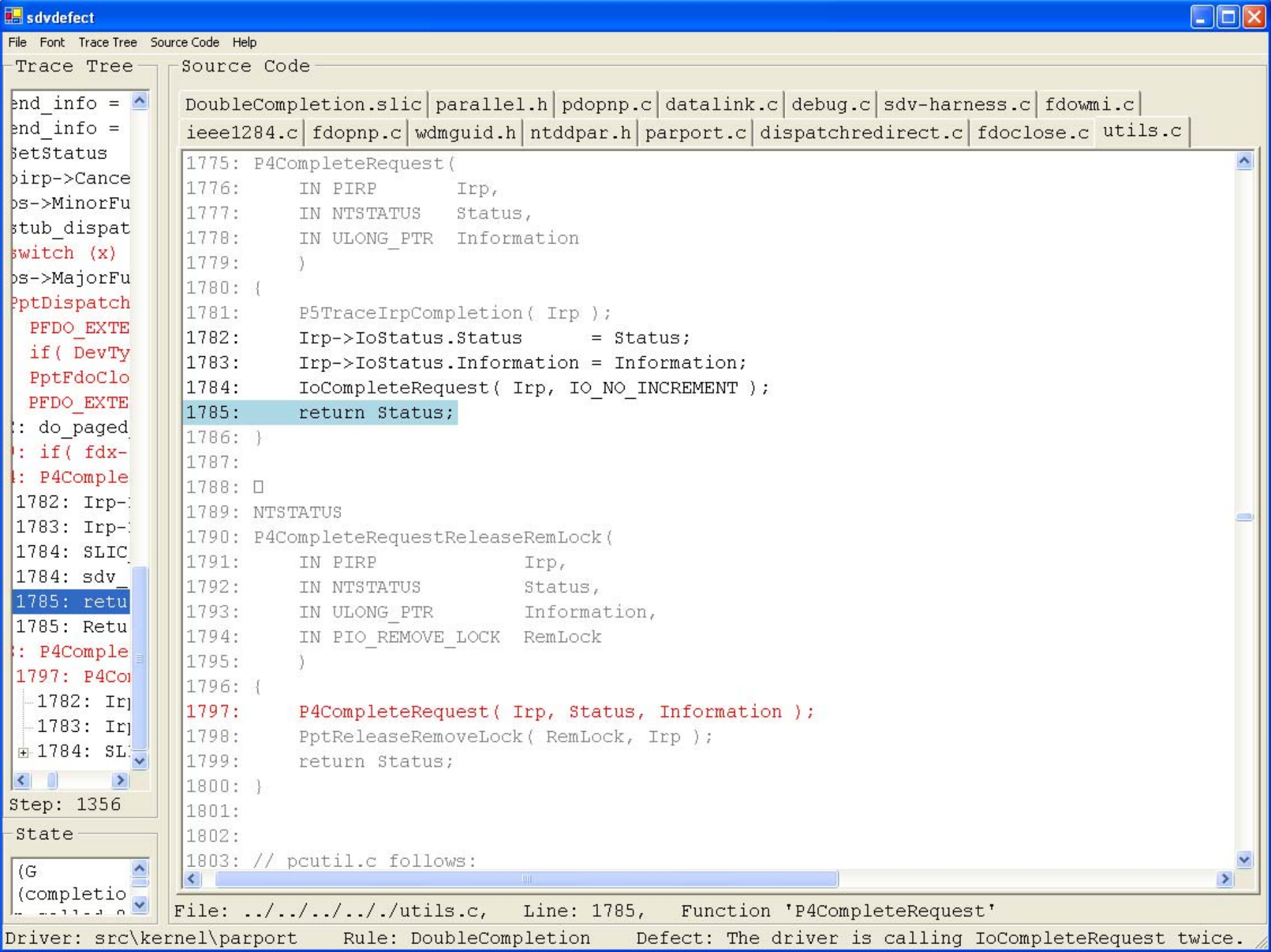

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
end_info
end_info
SetStatus
pirp->Can
ps->Minor
stub_disp
switch (x
ps->Major
PptDispat
5: PFDO_EX
7: if(Dev
8: PptFdoC
9: PFDO_EX
12: do_pag
19: if(fd
24: P4Comp
1782: Ir
1783: Ir
1784: SL
1784: sd
1785: re
1785: Re
63: P4Comp
1797: P4
1782:
1783:
1784:
Step: 1334
State
(!(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1: #include "pch.h"
2:
3: NTSTATUS
4: PptFdoClose(
5: IN PDEVICE_OBJECT DeviceObject,
6: IN PIRP Irp
7:)
8: {
9: PFDO_EXTENSION fdx = DeviceObject->DeviceExtension;
10: NTSTATUS status;
11:
12: PAGED_CODE();
13:
14: //
15: // Verify that our device has not been SUPRISE_REMOVED. Generally
16: // only parallel ports on hot-plug busses (e.g., PCMCIA) and
17: // parallel ports in docking stations will be surprise removed.
18: //
19: if(fdx->PnpState & PPT_DEVICE_SURPRISE_REMOVED) {
20: //
21: // Our device has been SURPRISE removed, but since this is only
22: // a CLOSE, SUCCEED anyway.
23: //
24: status = P4CompleteRequest(Irp, STATUS_SUCCESS, 0);
25:
26: goto target_exit;
27: }
28:
29:
File: ../../../.././fdoclose.c, Line: 19, Function 'PptFdoClose'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

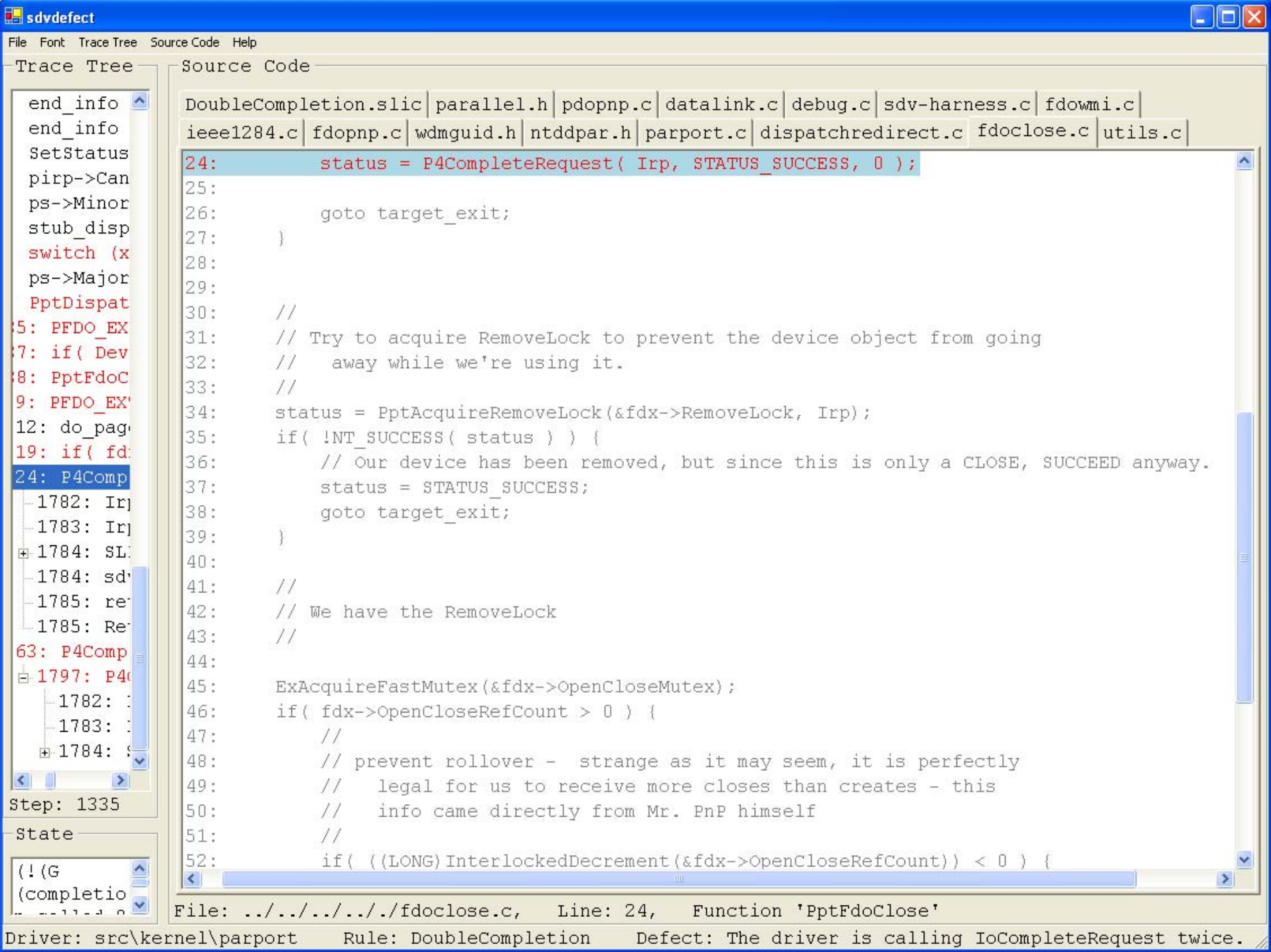

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
end_info
end_info
SetStatus
pirp->Can
ps->Minor
stub_disp
switch (x
ps->Major
PptDispat
5: PFDO_EX
7: if(Dev
8: PptFdoC
9: PFDO_EX'
12: do_pag
19: if(fd
24: P4Comp
1782: Irp
1783: Irp
1784: SL
1784: sd
1785: re
1785: Re
63: P4Comp
1797: P4
1782:
1783:
1784:
Step: 1335
State
(!(G
(completio
Source Code
DoubleCompletion.slic
parallel.h
pdopnp.c
datalink.c
debug.c
sdv-harness.c
fdowmi.c
ieee1284.c
fdopnp.c
wdmguid.h
ntddpar.h
parport.c
dispatchredirect.c
fdoclose.c
utils.c
2:
3: NTSTATUS
4: PptFdoClose(
5: IN PDEVICE_OBJECT DeviceObject,
6: IN PIRP Irp
7:)
8: {
9: PFDO_EXTENSION fdx = DeviceObject->DeviceExtension;
10: NTSTATUS status;
11:
12: PAGED_CODE();
13:
14: //
15: // Verify that our device has not been SUPRISE_REMOVED. Generally
16: // only parallel ports on hot-plug busses (e.g., PCMCIA) and
17: // parallel ports in docking stations will be surprise removed.
18: //
19: if(fdx->PnpState & PPT_DEVICE_SURPRISE_REMOVED) {
20: //
21: // Our device has been SURPRISE removed, but since this is only
22: // a CLOSE, SUCCEED anyway.
23: //
24: status = P4CompleteRequest(Irp, STATUS_SUCCESS, 0);
25:
26: goto target_exit;
27: }
28:
29:
30: //
File: ../../../.././fdoclose.c, Line: 24, Function 'PptFdoClose'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

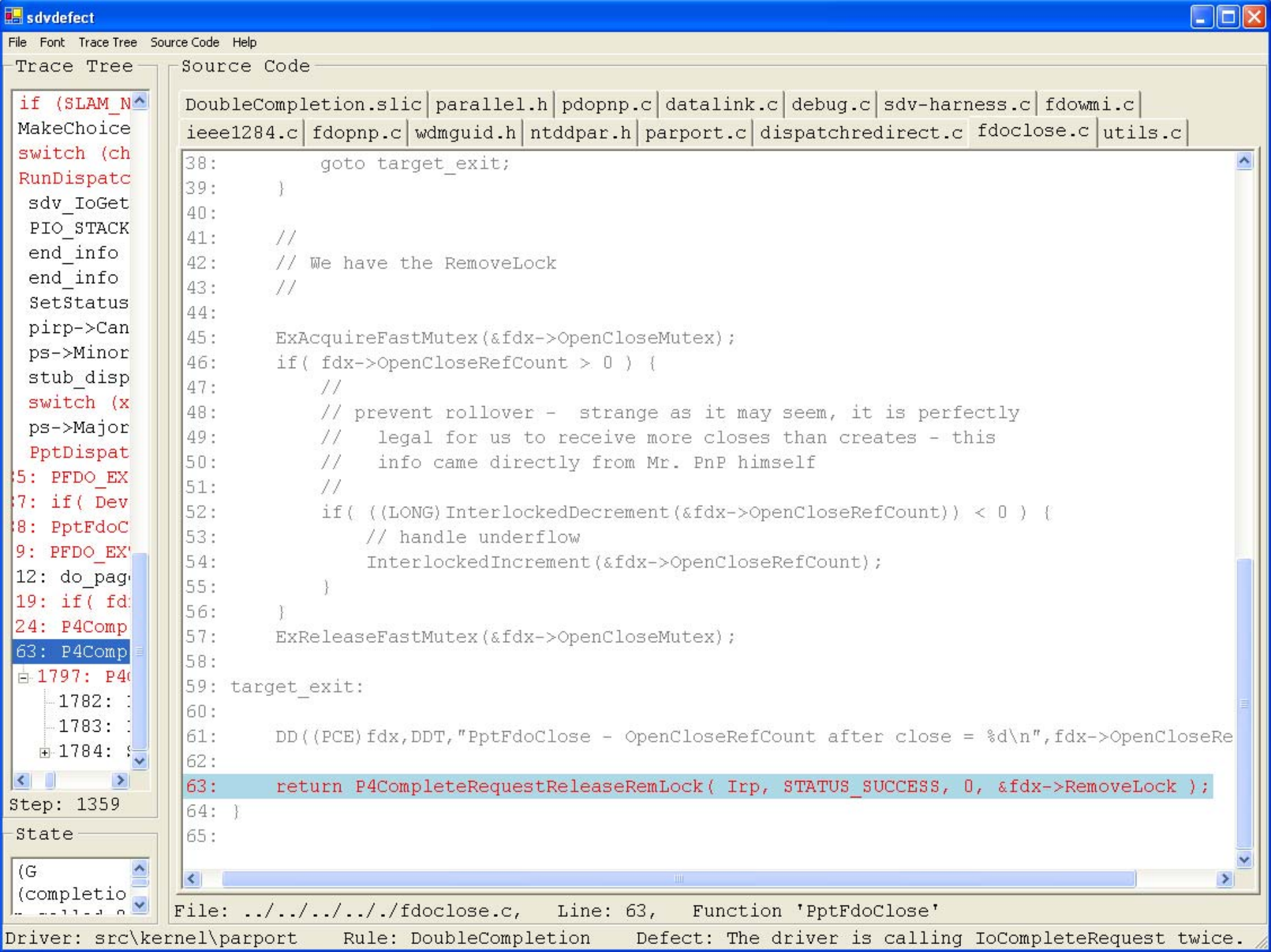

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
end_info =
end_info =
SetStatus
pirp->Cance
ps->MinorFu
stub_dispat
switch (x)
ps->MajorFu
PptDispatch
PFDO_EXTE
if(DevTy
PptFdoClo
PFDO_EXTE
: do_paged
: if(fdx-
: P4Comple
1782: Irp-
1783: Irp-
1784: SLIC
1784: sdv_
1785: retu
1785: Retu
: P4Comple
1797: P4Co
1782: Irj
1783: Irj
1784: SL
Step: 1338
State
(!(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c |
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1775: P4CompleteRequest(
1776: IN PIRP Irp,
1777: IN NTSTATUS Status,
1778: IN ULONG_PTR Information
1779:)
1780: {
1781: P5TraceIrpCompletion(Irp);
1782: Irp->IoStatus.Status = Status;
1783: Irp->IoStatus.Information = Information;
1784: IoCompleteRequest(Irp, IO_NO_INCREMENT);
1785: return Status;
1786: }
1787:
1788:
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791: IN PIRP Irp,
1792: IN NTSTATUS Status,
1793: IN ULONG_PTR Information,
1794: IN PIO_REMOVE_LOCK RemLock
1795:)
1796: {
1797: P4CompleteRequest(Irp, Status, Information);
1798: PptReleaseRemoveLock(RemLock, Irp);
1799: return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
File: ../../../.././utils.c, Line: 1782, Function 'P4CompleteRequest'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

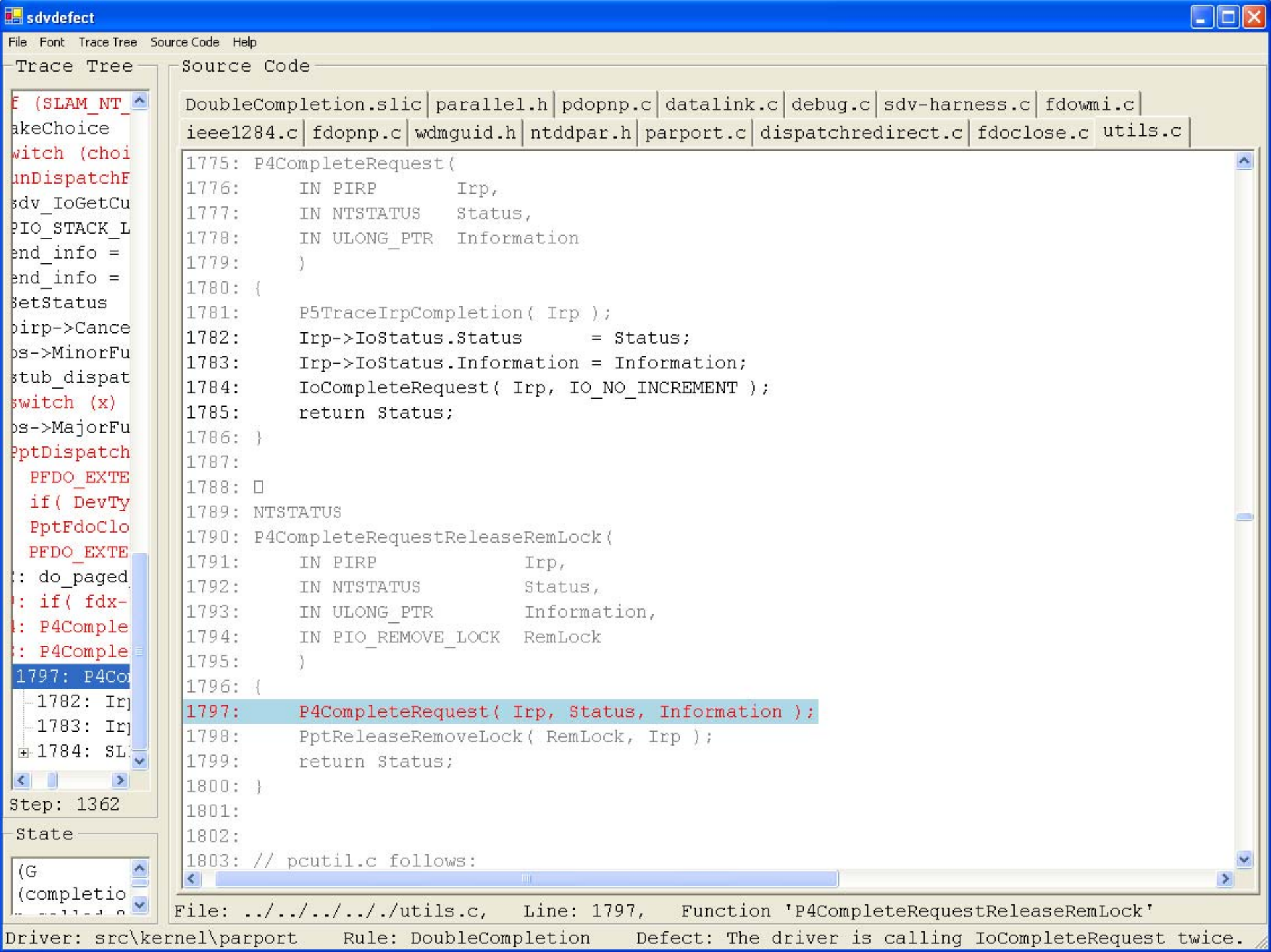

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
end_info =
end_info =
SetStatus
pirp->Cance
ps->MinorFu
stub_dispat
switch (x)
ps->MajorFu
PptDispatch
PFDO_EXTE
if(DevTy
PptFdoClo
PFDO_EXTE
: do_paged
: if(fdx-
: P4Comple
1782: Irp-
1783: Irp-
1784: SLIC
1784: sdv_
1785: retu
1785: Retu
: P4Comple
1797: P4Co
1782: Irj
1783: Irj
1784: SL
Step: 1339
State
(!(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1775: P4CompleteRequest(
1776: IN PIRP Irp,
1777: IN NTSTATUS Status,
1778: IN ULONG_PTR Information
1779:)
1780: {
1781: P5TraceIrpCompletion(Irp);
1782: Irp->IoStatus.Status = Status;
1783: Irp->IoStatus.Information = Information;
1784: IoCompleteRequest(Irp, IO_NO_INCREMENT);
1785: return Status;
1786: }
1787:
1788:
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791: IN PIRP Irp,
1792: IN NTSTATUS Status,
1793: IN ULONG_PTR Information,
1794: IN PIO_REMOVE_LOCK RemLock
1795:)
1796: {
1797: P4CompleteRequest(Irp, Status, Information);
1798: PptReleaseRemoveLock(RemLock, Irp);
1799: return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
File: ../../../.././utils.c, Line: 1783, Function 'P4CompleteRequest'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

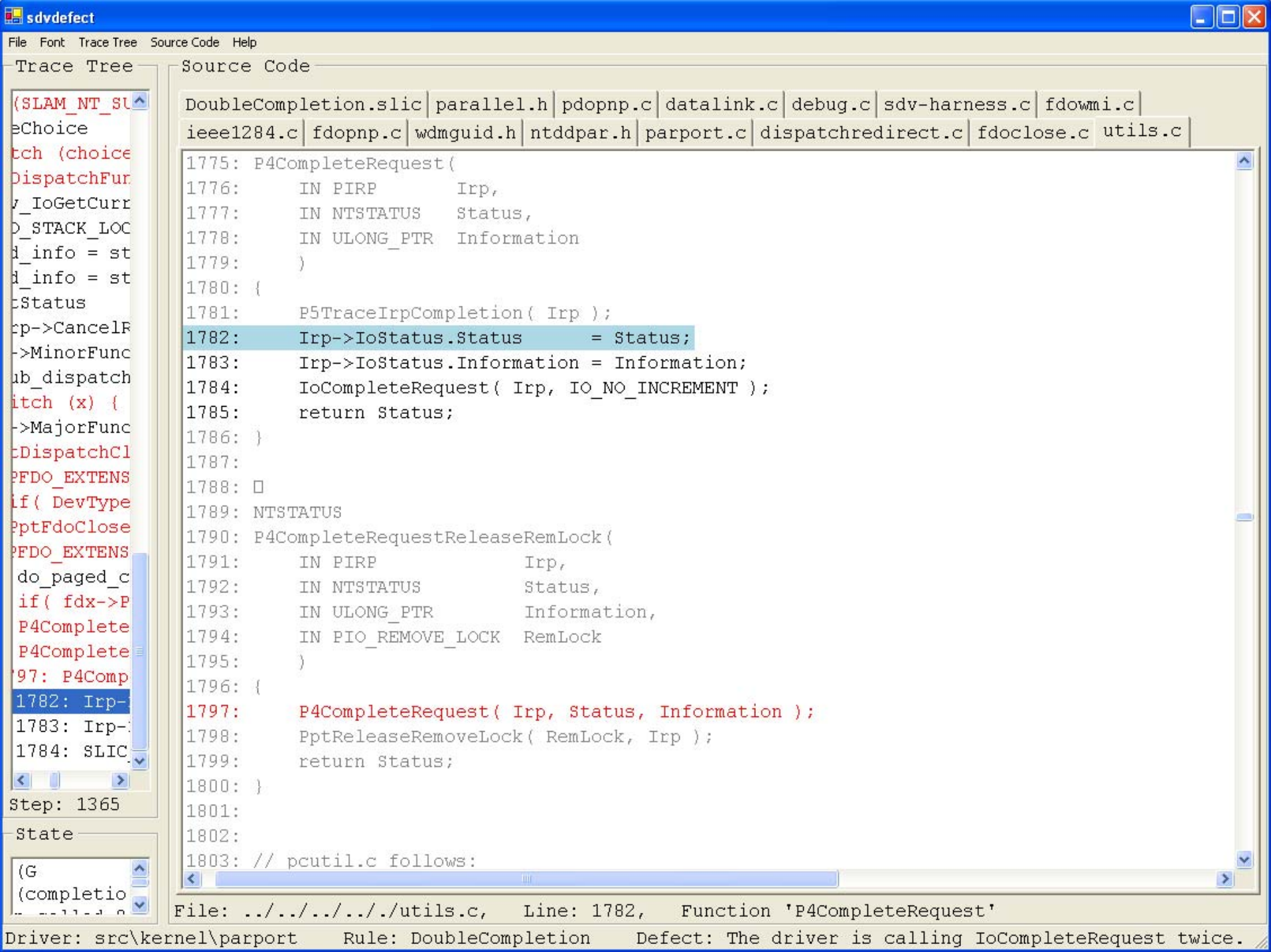

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
end_info =
end_info =
SetStatus
pirp->Cance
ps->MinorFu
stub_dispat
switch (x)
ps->MajorFu
PptDispatch
PFDO_EXTE
if(DevTy
PptFdoClo
PFDO_EXTE
: do_paged
: if(fdx-
: P4Comple
1782: Irp-
1783: Irp-
1784: SLIC
1784: sdv_
1785: retu
1785: Retu
: P4Comple
1797: P4Co
1782: Irj
1783: Irj
1784: SL
Step: 1341
State
(!(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1775: P4CompleteRequest(
1776: IN PIRP Irp,
1777: IN NTSTATUS Status,
1778: IN ULONG_PTR Information
1779:)
1780: {
1781: P5TraceIrpCompletion(Irp);
1782: Irp->IoStatus.Status = Status;
1783: Irp->IoStatus.Information = Information;
1784: IoCompleteRequest(Irp, IO_NO_INCREMENT);
1785: return Status;
1786: }
1787:
1788:
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791: IN PIRP Irp,
1792: IN NTSTATUS Status,
1793: IN ULONG_PTR Information,
1794: IN PIO_REMOVE_LOCK RemLock
1795:)
1796: {
1797: P4CompleteRequest(Irp, Status, Information);
1798: PptReleaseRemoveLock(RemLock, Irp);
1799: return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
File: ../../../.././utils.c, Line: 1784, Function 'P4CompleteRequest'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

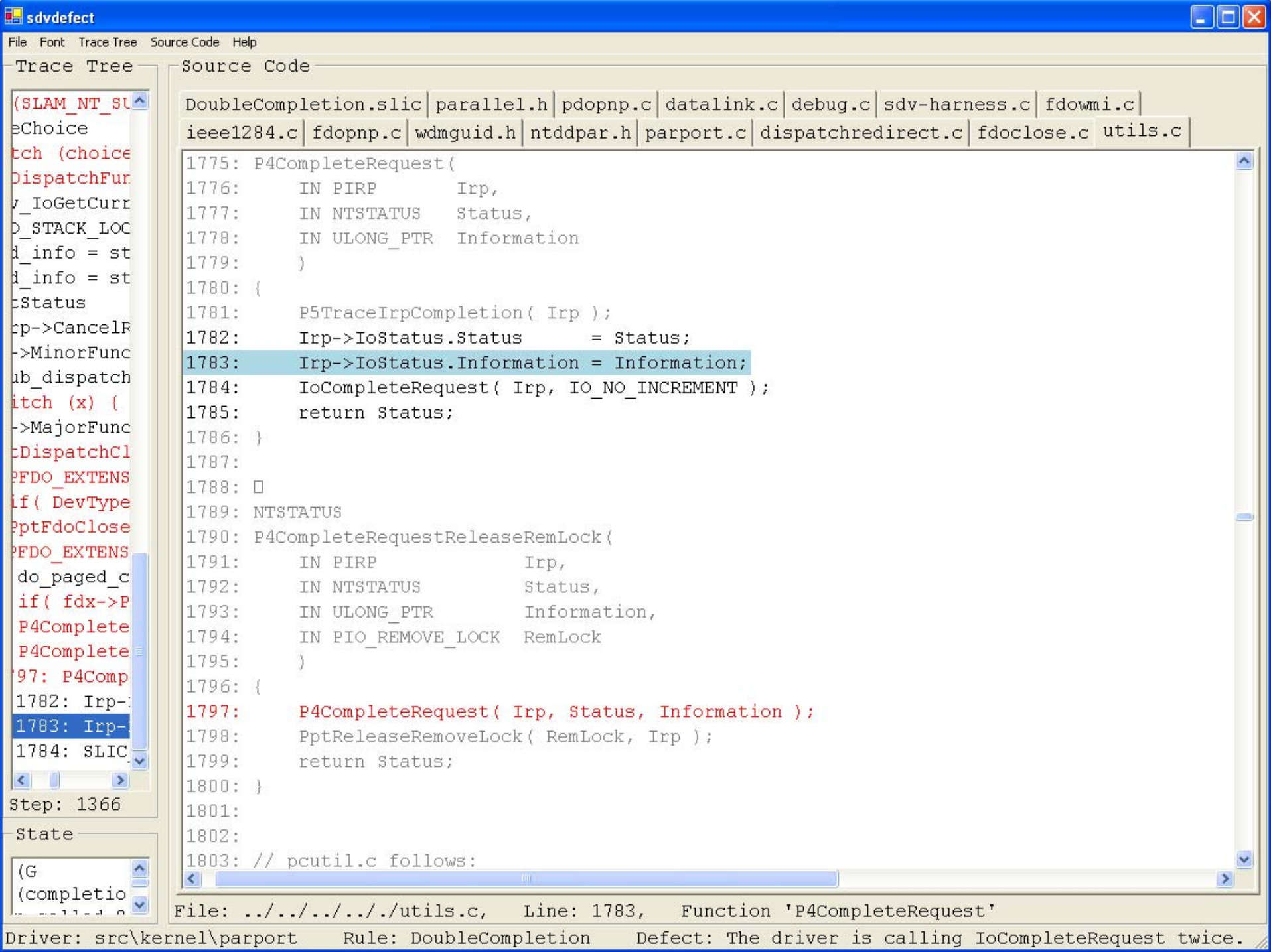

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
end_info =
end_info =
SetStatus
pirp->Cance
ps->MinorFu
stub_dispat
switch (x)
ps->MajorFu
PptDispatch
PFDO_EXTE
if(DevTy
PptFdoClo
PFDO_EXTE
: do_paged
: if(fdx-
: P4Comple
1782: Irp-
1783: Irp-
1784: SLIC
1784: sdv_
1785: retu
1785: Retu
: P4Comple
1797: P4Co
1782: Irj
1783: Irj
1784: SL
Step: 1356
State
(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1775: P4CompleteRequest(
1776: IN PIRP Irp,
1777: IN NTSTATUS Status,
1778: IN ULONG_PTR Information
1779:)
1780: {
1781: P5TraceIrpCompletion(Irp);
1782: Irp->IoStatus.Status = Status;
1783: Irp->IoStatus.Information = Information;
1784: IoCompleteRequest(Irp, IO_NO_INCREMENT);
1785: return Status;
1786: }
1787:
1788:
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791: IN PIRP Irp,
1792: IN NTSTATUS Status,
1793: IN ULONG_PTR Information,
1794: IN PIO_REMOVE_LOCK RemLock
1795:)
1796: {
1797: P4CompleteRequest(Irp, Status, Information);
1798: PptReleaseRemoveLock(RemLock, Irp);
1799: return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
File: ../../../.././utils.c, Line: 1785, Function 'P4CompleteRequest'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

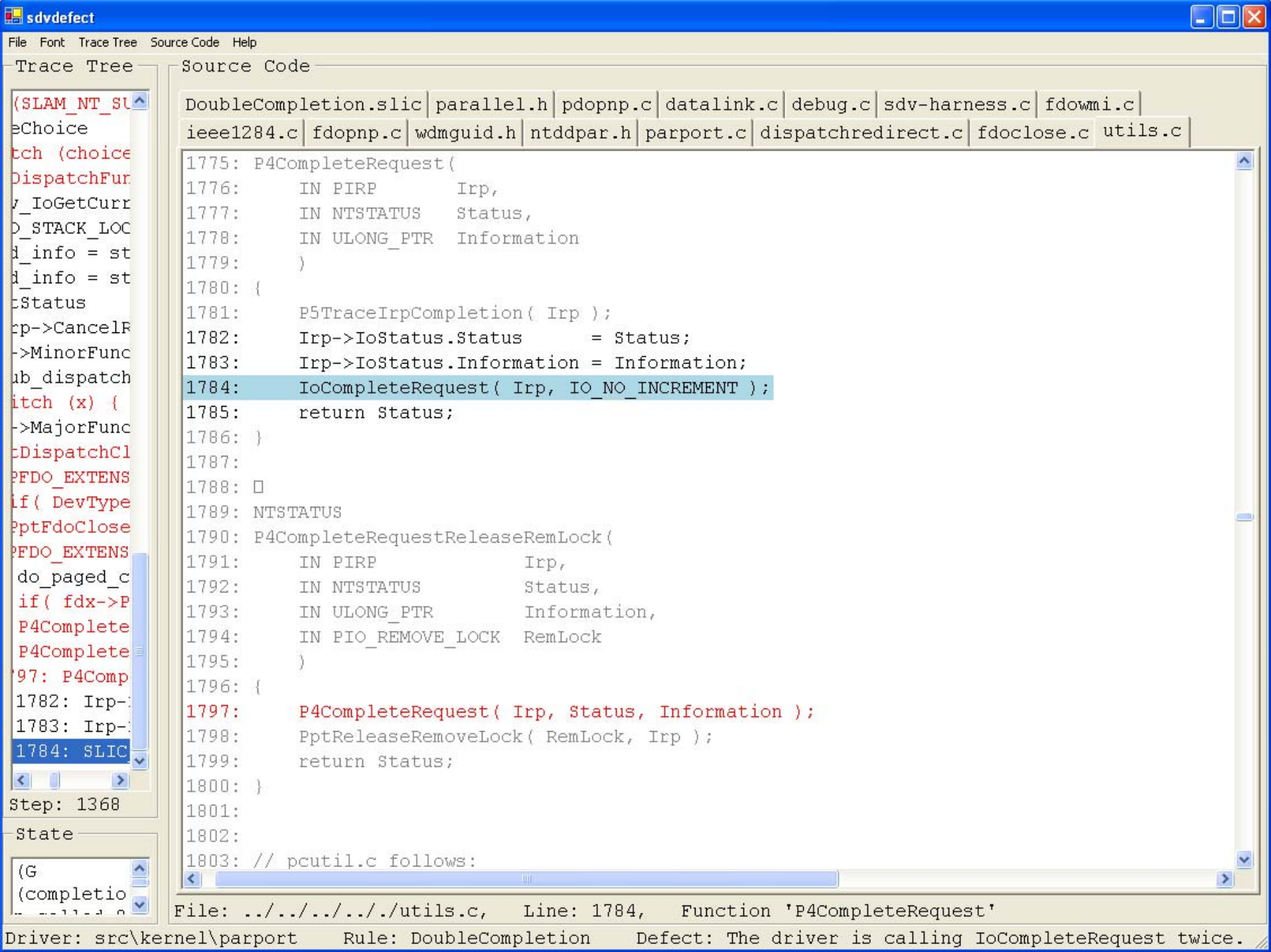

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
end_info
end_info
SetStatus
pirp->Can
ps->Minor
stub_disp
switch (x
ps->Major
PptDispat
5: PFDO_EX
7: if(Dev
8: PptFdoC
9: PFDO_EX
12: do_pag
19: if(fd
24: P4Comp
1782: Ir
1783: Ir
1784: SL
1784: sd
1785: re
1785: Re
63: P4Comp
1797: P4
1782:
1783:
1784:
Step: 1335
State
(!(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
24: status = P4CompleteRequest(Irp, STATUS_SUCCESS, 0);
25:
26: goto target_exit;
27: }
28:
29:
30: //
31: // Try to acquire RemoveLock to prevent the device object from going
32: // away while we're using it.
33: //
34: status = PptAcquireRemoveLock(&fdx->RemoveLock, Irp);
35: if(!NT_SUCCESS(status)) {
36: // Our device has been removed, but since this is only a CLOSE, SUCCEED anyway.
37: status = STATUS_SUCCESS;
38: goto target_exit;
39: }
40:
41: //
42: // We have the RemoveLock
43: //
44:
45: ExAcquireFastMutex(&fdx->OpenCloseMutex);
46: if(fdx->OpenCloseRefCount > 0) {
47: //
48: // prevent rollover - strange as it may seem, it is perfectly
49: // legal for us to receive more closes than creates - this
50: // info came directly from Mr. PnP himself
51: //
52: if(((LONG)InterlockedDecrement(&fdx->OpenCloseRefCount)) < 0) {
File: ../../../.././fdoclose.c, Line: 24, Function 'PptFdoClose'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.


sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
if (SLAM_N
MakeChoice
switch (ch
RunDispatc
sdv_IoGet
PIO_STACK
end_info
end_info
SetStatus
pirp->Can
ps->Minor
stub_disp
switch (x
ps->Major
PptDispat
5: PFDO_EX
7: if(Dev
8: PptFdoC
9: PFDO_EX
12: do_pag
19: if(fd
24: P4Comp
63: P4Comp
1797: P4
1782:
1783:
1784:
Step: 1359
State
(G
(completio
Source Code
DoubleCompletion.slic
parallel.h
pdopnp.c
datalink.c
debug.c
sdv-harness.c
fdowmi.c
ieee1284.c
fdopnp.c
wdmguid.h
ntddpar.h
parport.c
dispatchredirect.c
fdoclose.c
utils.c
38: goto target_exit;
39: }
40:
41: //
42: // We have the RemoveLock
43: //
44:
45: ExAcquireFastMutex(&fdx->OpenCloseMutex);
46: if(fdx->OpenCloseRefCount > 0) {
47: //
48: // prevent rollover - strange as it may seem, it is perfectly
49: // legal for us to receive more closes than creates - this
50: // info came directly from Mr. PnP himself
51: //
52: if(((LONG)InterlockedDecrement(&fdx->OpenCloseRefCount)) < 0) {
53: // handle underflow
54: InterlockedIncrement(&fdx->OpenCloseRefCount);
55: }
56: }
57: ExReleaseFastMutex(&fdx->OpenCloseMutex);
58:
59: target_exit:
60:
61: DD((PCE)fdx,DDT,"PptFdoClose - OpenCloseRefCount after close = %d\n",fdx->OpenCloseRe
62:
63: return P4CompleteRequestReleaseRemLock(Irp, STATUS_SUCCESS, 0, &fdx->RemoveLock);
64: }
65:
File: ../../../.././fdoclose.c, Line: 63, Function 'PptFdoClose'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.


sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1775: P4CompleteRequest(
1776: IN PIRP Irp,
1777: IN NTSTATUS Status,
1778: IN ULONG_PTR Information
1779:)
1780: {
1781: P5TraceIrpCompletion(Irp);
1782: Irp->IoStatus.Status = Status;
1783: Irp->IoStatus.Information = Information;
1784: IoCompleteRequest(Irp, IO_NO_INCREMENT);
1785: return Status;
1786: }
1787:
1788:
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791: IN PIRP Irp,
1792: IN NTSTATUS Status,
1793: IN ULONG_PTR Information,
1794: IN PIO_REMOVE_LOCK RemLock
1795:)
1796: {
1797: P4CompleteRequest(Irp, Status, Information);
1798: PptReleaseRemoveLock(RemLock, Irp);
1799: return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
Step: 1362
State
File: ../../../.././utils.c, Line: 1797, Function 'P4CompleteRequestReleaseRemLock'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.


sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
(SLAM_NT_SU
eChoice
tch (choice
DispatchFur
v_IoGetCurr
O_STACK_LOC
d_info = st
d_info = st
tStatus
cp->CancelR
->MinorFunc
ub_dispatch
itch (x) {
->MajorFunc
tDispatchCl
PFDO_EXTENS
if(DevType
PptFdoClose
PFDO_EXTENS
do_paged_c
if(fdx->P
P4Complete
P4Complete
97: P4Comp
1782: Irp-
1783: Irp-
1784: SLIC
Step: 1365
State
(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1775: P4CompleteRequest(
1776: IN PIRP Irp,
1777: IN NTSTATUS Status,
1778: IN ULONG_PTR Information
1779:)
1780: {
1781: P5TraceIrpCompletion(Irp);
1782: Irp->IoStatus.Status = Status;
1783: Irp->IoStatus.Information = Information;
1784: IoCompleteRequest(Irp, IO_NO_INCREMENT);
1785: return Status;
1786: }
1787:
1788:
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791: IN PIRP Irp,
1792: IN NTSTATUS Status,
1793: IN ULONG_PTR Information,
1794: IN PIO_REMOVE_LOCK RemLock
1795:)
1796: {
1797: P4CompleteRequest(Irp, Status, Information);
1798: PptReleaseRemoveLock(RemLock, Irp);
1799: return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
File: ../../../.././utils.c, Line: 1782, Function 'P4CompleteRequest'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

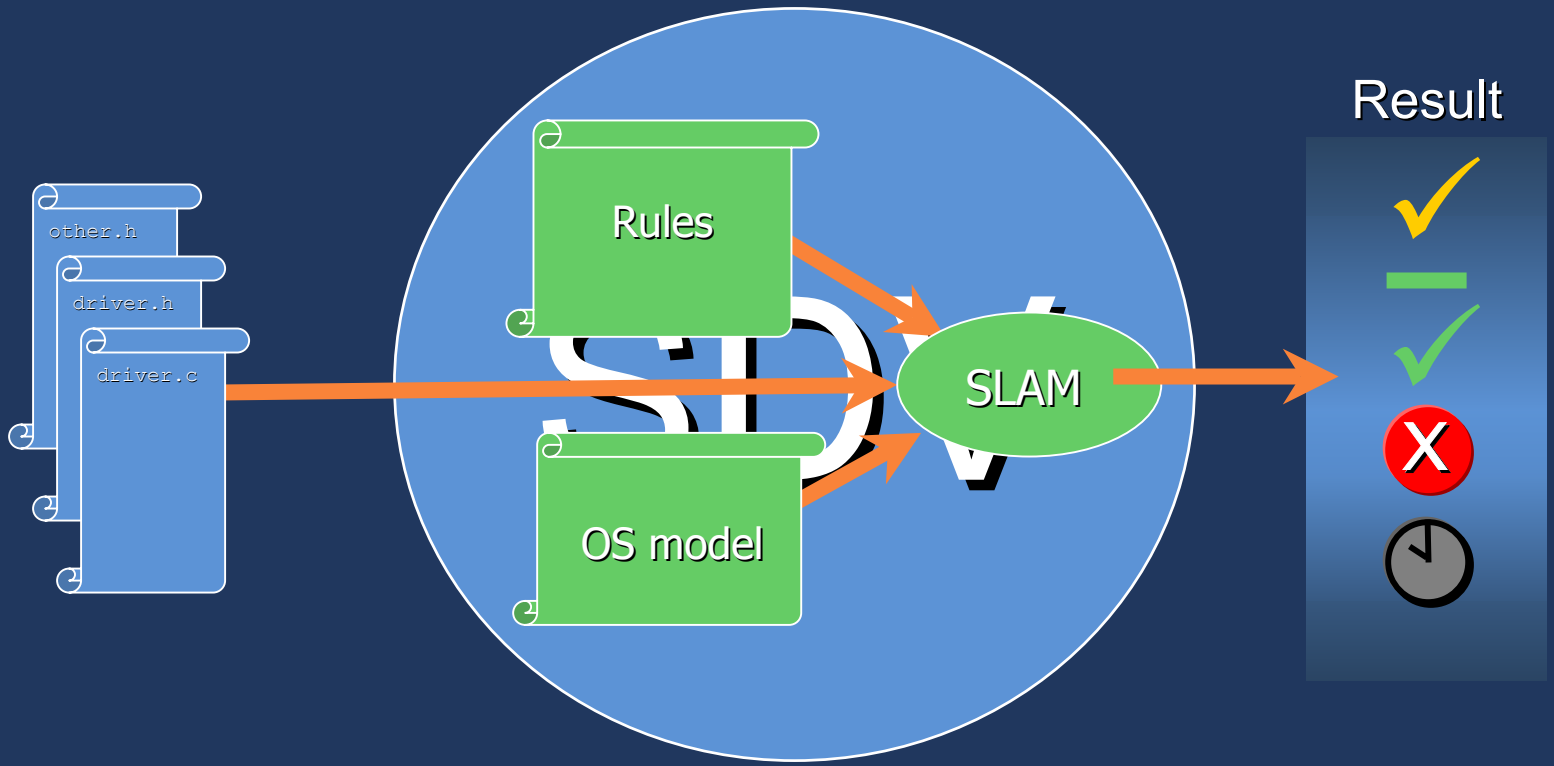

sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
(SLAM_NT_SU
eChoice
tch (choice
DispatchFur
v_IoGetCurr
O_STACK_LOC
d_info = st
d_info = st
tStatus
cp->CancelR
->MinorFunc
ub_dispatch
itch (x) {
->MajorFunc
tDispatchCl
PFDO_EXTENS
if(DevType
PptFdoClose
PFDO_EXTENS
do_paged_c
if(fdx->P
P4Complete
P4Complete
97: P4Comp
1782: Irp-
1783: Irp-
1784: SLIC
Step: 1366
State
(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1775: P4CompleteRequest(
1776: IN PIRP Irp,
1777: IN NTSTATUS Status,
1778: IN ULONG_PTR Information
1779:)
1780: {
1781: P5TraceIrpCompletion(Irp);
1782: Irp->IoStatus.Status = Status;
1783: Irp->IoStatus.Information = Information;
1784: IoCompleteRequest(Irp, IO_NO_INCREMENT);
1785: return Status;
1786: }
1787:
1788:
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791: IN PIRP Irp,
1792: IN NTSTATUS Status,
1793: IN ULONG_PTR Information,
1794: IN PIO_REMOVE_LOCK RemLock
1795:)
1796: {
1797: P4CompleteRequest(Irp, Status, Information);
1798: PptReleaseRemoveLock(RemLock, Irp);
1799: return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
File: ../../../.././utils.c, Line: 1783, Function 'P4CompleteRequest'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.


sdvdefect
File Font Trace Tree Source Code Help
Trace Tree
(SLAM_NT_SU
eChoice
tch (choice
DispatchFur
v_IoGetCurr
O_STACK_LOC
d_info = st
d_info = st
tStatus
cp->CancelR
->MinorFunc
ub_dispatch
itch (x) {
->MajorFunc
tDispatchCl
PFDO_EXTENS
if(DevType
PptFdoClose
PFDO_EXTENS
do_paged_c
if(fdx->P
P4Complete
P4Complete
97: P4Comp
1782: Irp-
1783: Irp-
1784: SLIC
Step: 1368
State
(G
(completio
Source Code
DoubleCompletion.slic | parallel.h | pdopnp.c | datalink.c | debug.c | sdv-harness.c | fdowmi.c
ieee1284.c | fdopnp.c | wdmguid.h | ntddpar.h | parport.c | dispatchredirect.c | fdoclose.c | utils.c
1775: P4CompleteRequest(
1776: IN PIRP Irp,
1777: IN NTSTATUS Status,
1778: IN ULONG_PTR Information
1779:)
1780: {
1781: P5TraceIrpCompletion(Irp);
1782: Irp->IoStatus.Status = Status;
1783: Irp->IoStatus.Information = Information;
1784: IoCompleteRequest(Irp, IO_NO_INCREMENT);
1785: return Status;
1786: }
1787:
1788:
1789: NTSTATUS
1790: P4CompleteRequestReleaseRemLock(
1791: IN PIRP Irp,
1792: IN NTSTATUS Status,
1793: IN ULONG_PTR Information,
1794: IN PIO_REMOVE_LOCK RemLock
1795:)
1796: {
1797: P4CompleteRequest(Irp, Status, Information);
1798: PptReleaseRemoveLock(RemLock, Irp);
1799: return Status;
1800: }
1801:
1802:
1803: // pcutil.c follows:
File: ../../../.././utils.c, Line: 1784, Function 'P4CompleteRequest'
Driver: src\kernel\parport Rule: DoubleCompletion Defect: The driver is calling IoCompleteRequest twice.

Introduction To Static Driver Verifier

- SDV is being deployed internally within Microsoft. Driver writers are giving us positive feedback
  - "This bug would be a really hard bug to find other than with a tool like SDV. There are just too many details to keep track of to have a good chance of finding it."
  - "This looks like a bug to me. This is AWESOME!! SDV rocks!"
  - "These are all real, difficult to discover bugs. Good work!"
  - "This bug would have been very difficult to find by inspection and it was one of those bugs that would be near-impossible to reproduce…"
  - "Fixing this bug will definitely stop some unexplainable and hard to debug random system crashes in the future"
- SDV was used on 3rd party drivers at the Windows Driver Development Conference in November
  - We found some real bugs in these drivers
- We're planning to make SDV available externally, later this year

Static Driver Verifier Internals: Rules



Static Driver Verifier Internals: Rules

- Expressed in an event-based language called SLIC
- Possible events
  - Function entry
  - Function exit
- The code associated with events call the function **`error()`** to indicate a violation

```
IoCallDriver.entry
{
    if ($2->Tail.Overlay.CurrentStackLocation-
>MajorFunction
            ==IRP_MJ_POWER) {
        error();
    }
}
```

Static Driver Verifier Internals: Rules

```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}

stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```

Static Driver Verifier internals: Rules

```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}

stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```

Static Driver Verifier Internals: Rules

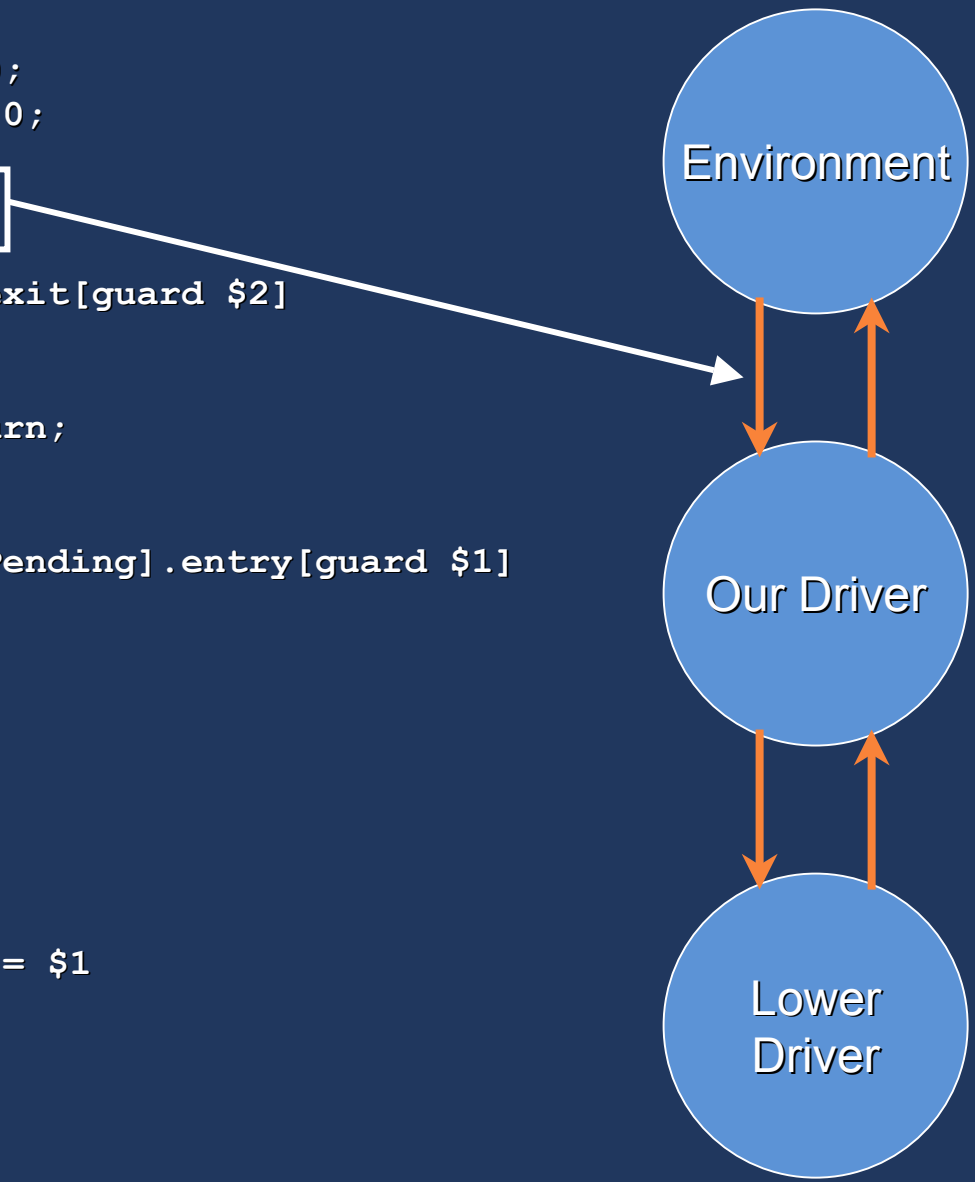
```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}

stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```

Static Driver Verifier Internals: Rules

```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}

stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```

Static Driver Verifier Internals: Rules

```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}

stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```

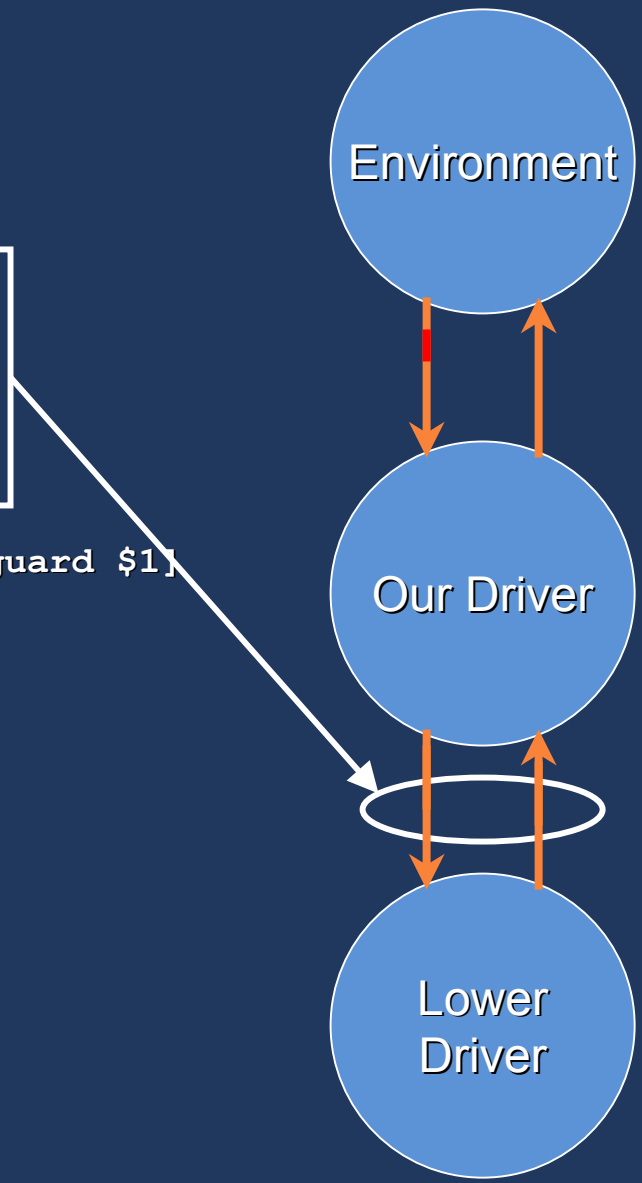


Static Driver Verifier Internals: Rules

```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}

stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```

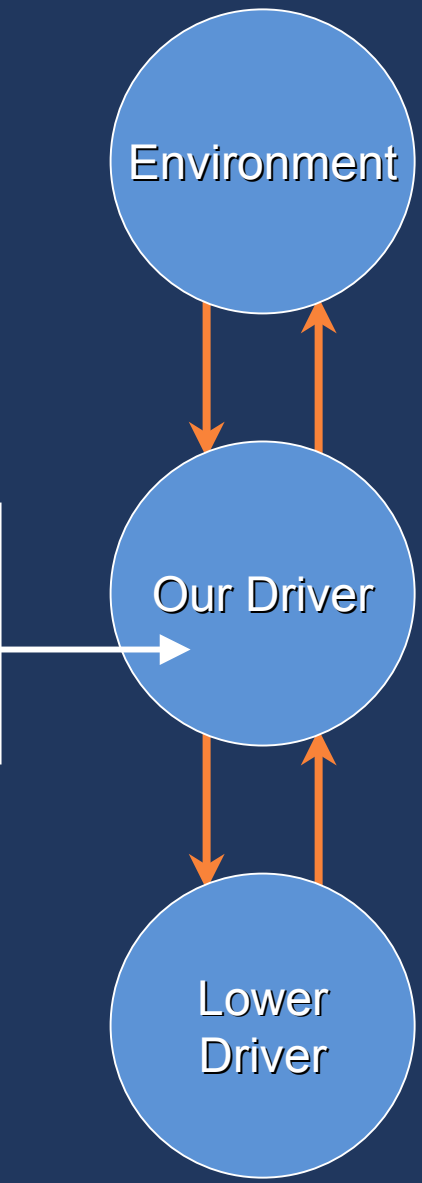
Static Driver Verifier Internals: Rules

```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}

stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```



Static Driver Verifier Internals: Rules

```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}

stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```

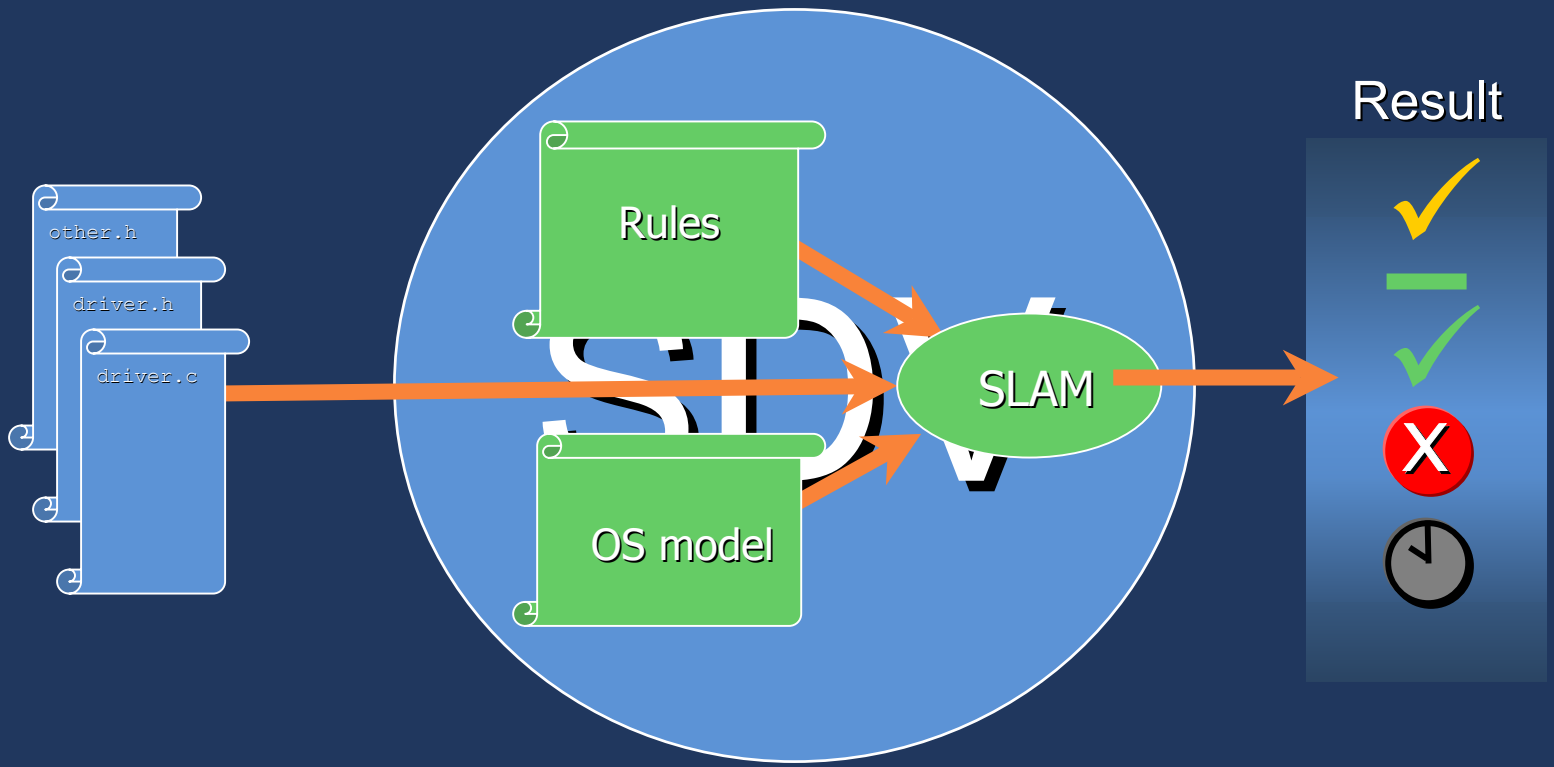


Static Driver Verifier Internals: Rules

```
state {
    int LowerDriverReturn = 0;
    bool LowerDriverCalled = 0;
    bool PerformCheck = 1;
} with guard (sdv_main,irp)

[PoCallDriver,IoCallDriver].exit[guard $2]
{
    LowerDriverCalled = 1;
    LowerDriverReturn = $return;
}

[IoCompleteRequest,IoMarkIrpPending].entry[guard $1]
{
    PerformCheck = 0;
}
```

```
stub_dispatch_end.entry
{
    if (  PerformCheck
       && LowerDriverCalled
       && LowerDriverReturn != $1
       ) {
          error "……";
    }
}
```

Static Driver Verifier Internals: OS Model



Static Driver Verifier Internals: OS Model

- Provides the main function
- Abstract implementations of kernel APIs (like IoCallDriver)
- Models some aspects of the OS state, like the “interrupt request level” (IRQL)
- Uses non-deterministic choice

Static Driver Verifier Internals: OS Model

```
VOID IoAcquireCancelSpinLock( OUT PKIRQL Irql )
{
    old_old_old_irql = old_old_irql;
    old_old_irql = old_irql;
    old_irql = irql;
    irql = DISPATCH_LEVEL;
    *Irql = old_irql;
}
```

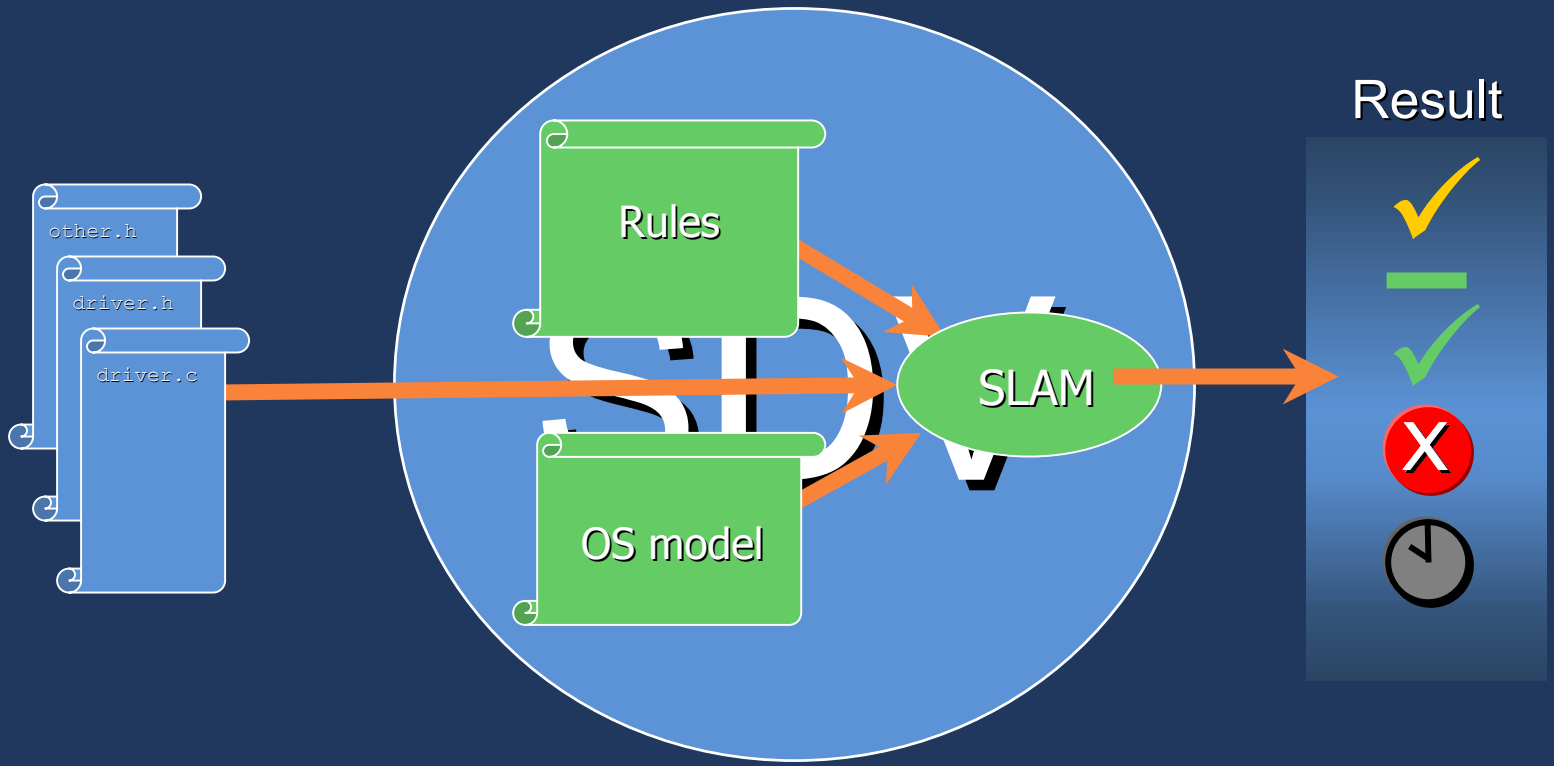
Static Driver Verifier Internals: OS Model

```
NTSTATUS
IoCreateDevice(
    IN PDRIVER_OBJECT DriverObject,
    IN ULONG DeviceExtensionSize,
    IN PUNICODE_STRING DeviceName OPTIONAL,
    IN DEVICE_TYPE DeviceType,
    IN ULONG DeviceCharacterisitcs,
    IN BOOLEAN Exclusive,
    OUT PDEVICE_OBJECT * DeviceObject
    )
{

    ULONG Choice = SdvChoice();

    switch (Choice) {
        case 0 : (*DeviceObject) = &SDV_devobj2;
                 return STATUS_SUCCESS;
        case 1 : return STATUS_INSUFFICIENT_RESOURCES;
        case 2 : return STATUS_OBJECT_NAME_EXISTS;
        default: return STATUS_OBJECT_NAME_COLLISION;
    }
}
```

Static Driver Verifier Internals: OS Model

```
NTSTATUS
IoCreateDevice(
    IN PDRIVER_OBJECT DriverObject,
    IN ULONG DeviceExtensionSize,
    IN PUNICODE_STRING DeviceName OPTIONAL,
    IN DEVICE_TYPE DeviceType,
    IN ULONG DeviceCharacterisitcs,
    IN BOOLEAN Exclusive,
    OUT PDEVICE_OBJECT * DeviceObject
    )
{

    ULONG Choice = SdvChoice();

    switch (Choice) {
        case 0 : (*DeviceObject) = &SDV_devobj2;
                 return STATUS_SUCCESS;
        case 1 : return STATUS_INSUFFICIENT_RESOURCES;
        case 2 : return STATUS_OBJECT_NAME_EXISTS;
        default: return STATUS_OBJECT_NAME_COLLISION;
    }
}
```

Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM

- Proof-based analysis engine for C (not C++)
- Strategy:
  - Build an abstraction of the driver
  - throw away as much irrelevant detail from the driver as possible through abstraction search
- Simplifying (unsound) assumptions
  - C unions are ignored
  - Memory layout is not known: pointer arithmetic is largely ignored
  - Coincidental pointer aliasing is ignored, purposeful aliasing is not
  - The OS model does not exercise all paths possible in practice
  - Etc, etc.

Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
         }
         g();
    }
}
```

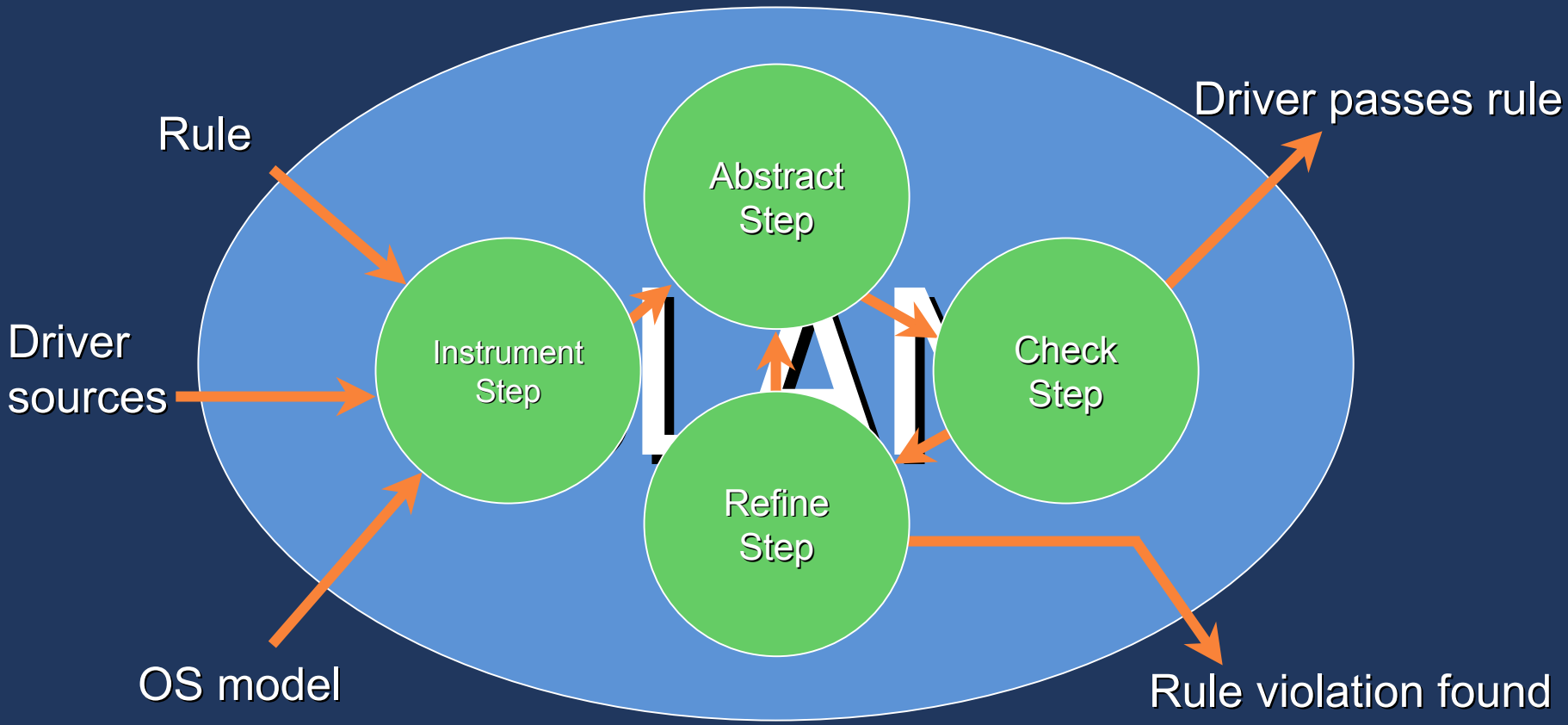
Assume that **f1**, **f2**, **f3** and **g** do not call **AcquireLock** or **ReleaseLock**

Static Driver Verifier Internals: SLAM

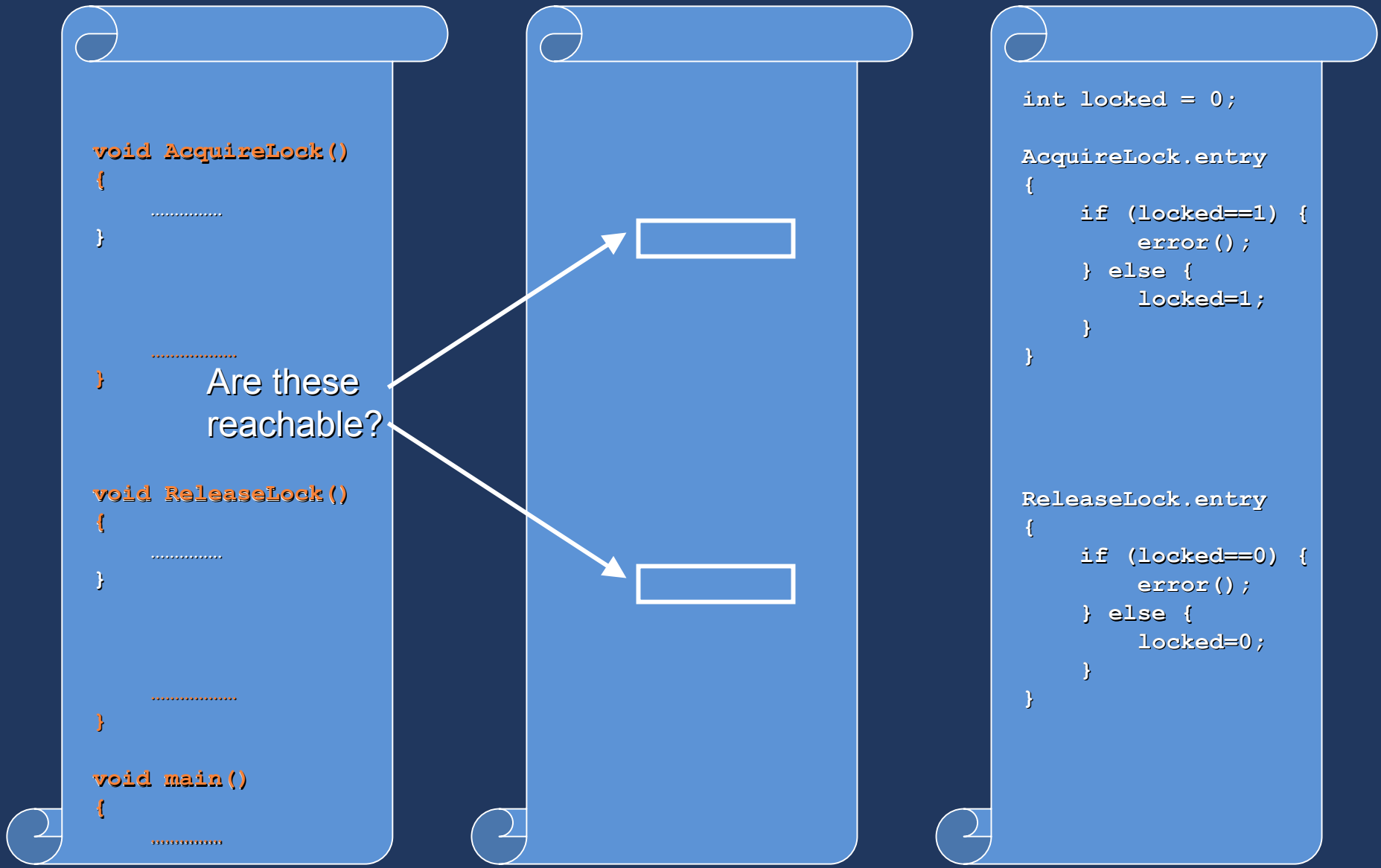
```
int locked = 0;


AcquireLock.entry {
    if (locked==1) {
        error();
    } else {
        locked=1;
    }
}

ReleaseLock.entry {
    if (locked==0) {
        error();
    } else {
        locked=0;
    }
}
```

Static Driver Verifier Internals: SLAM

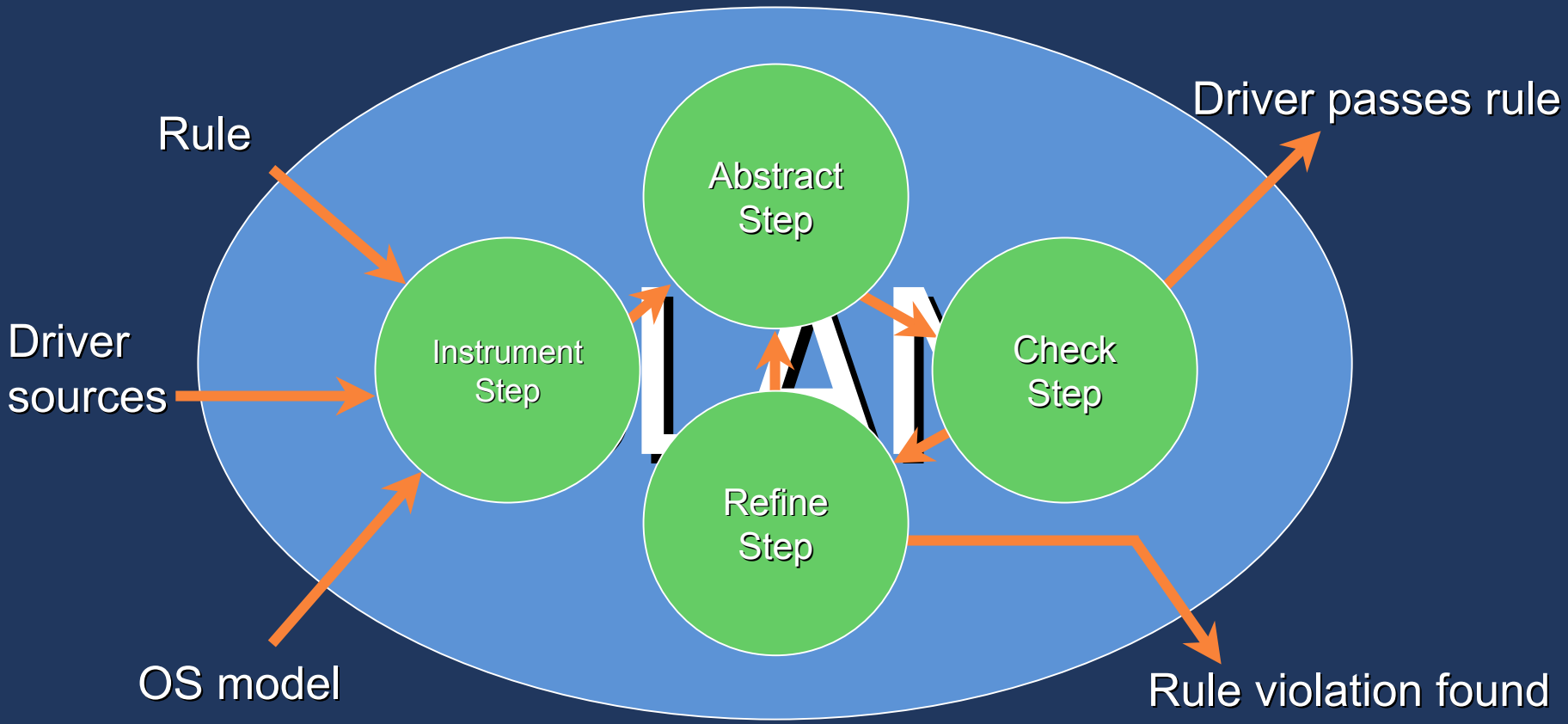
Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst ) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c ) {
                rst=1;
                ReleaseLock();
            }
         }
         g();
    }
}
```

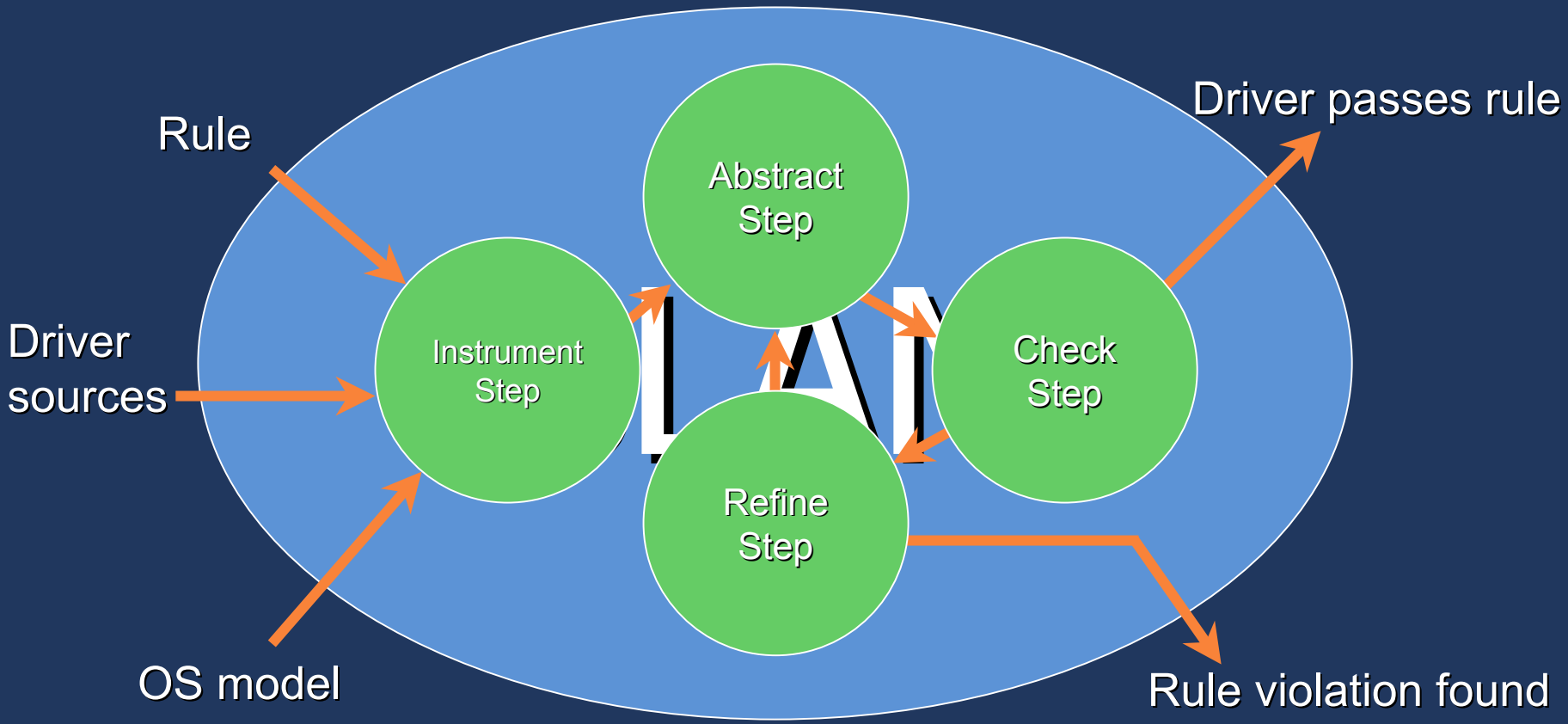
Static Driver Verifier Internals: SLAM

```
int locked = 0;
void AcquireLock()
{
    if (locked==1) {
        error();
    } else {
        locked = 1;
    }
}
void ReleaseLock()
{
    if (locked==0) {
        error();
    } else {
        locked = 0;
    }
}
```

```
bool g0 = 0; // locked==0
bool g1 = 0; // locked==1
void AcquireLock()
{
    if (g1==1) {
        error();
    } else {
        g1 = 1;
        g0 = 0;
    }
}
void ReleaseLock()
{
    if (g0==0) {
        error();
    } else {
        g0 = 1;
        g1 = 0;
    }
}
```

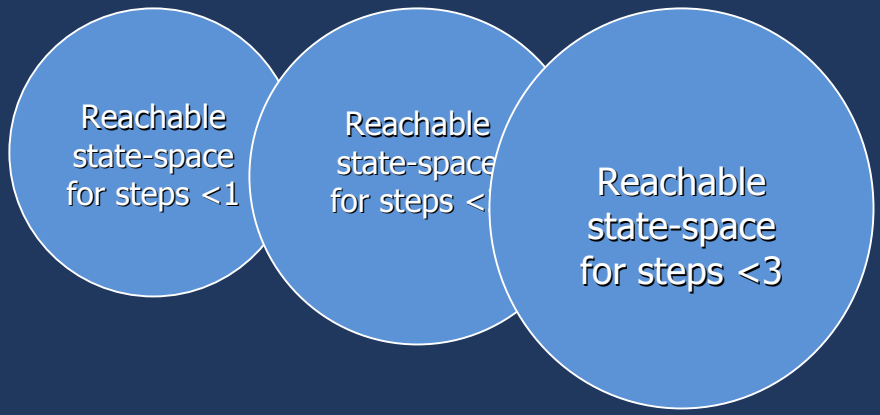
State space = 2^(2 + #bits(pc)) + stack

Static Driver Verifier Internals: SLAM

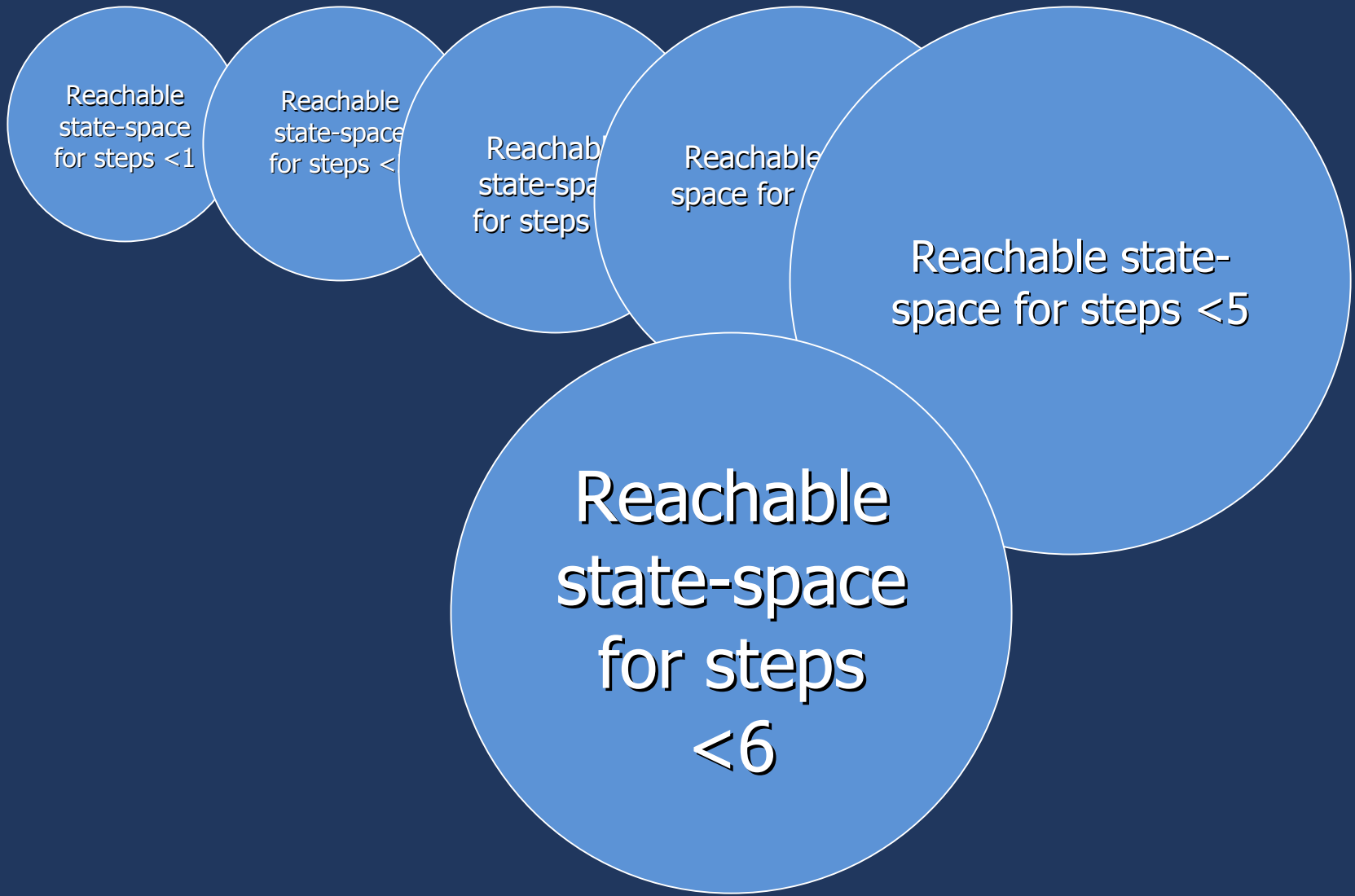


Static Driver Verifier Internals: SLAM

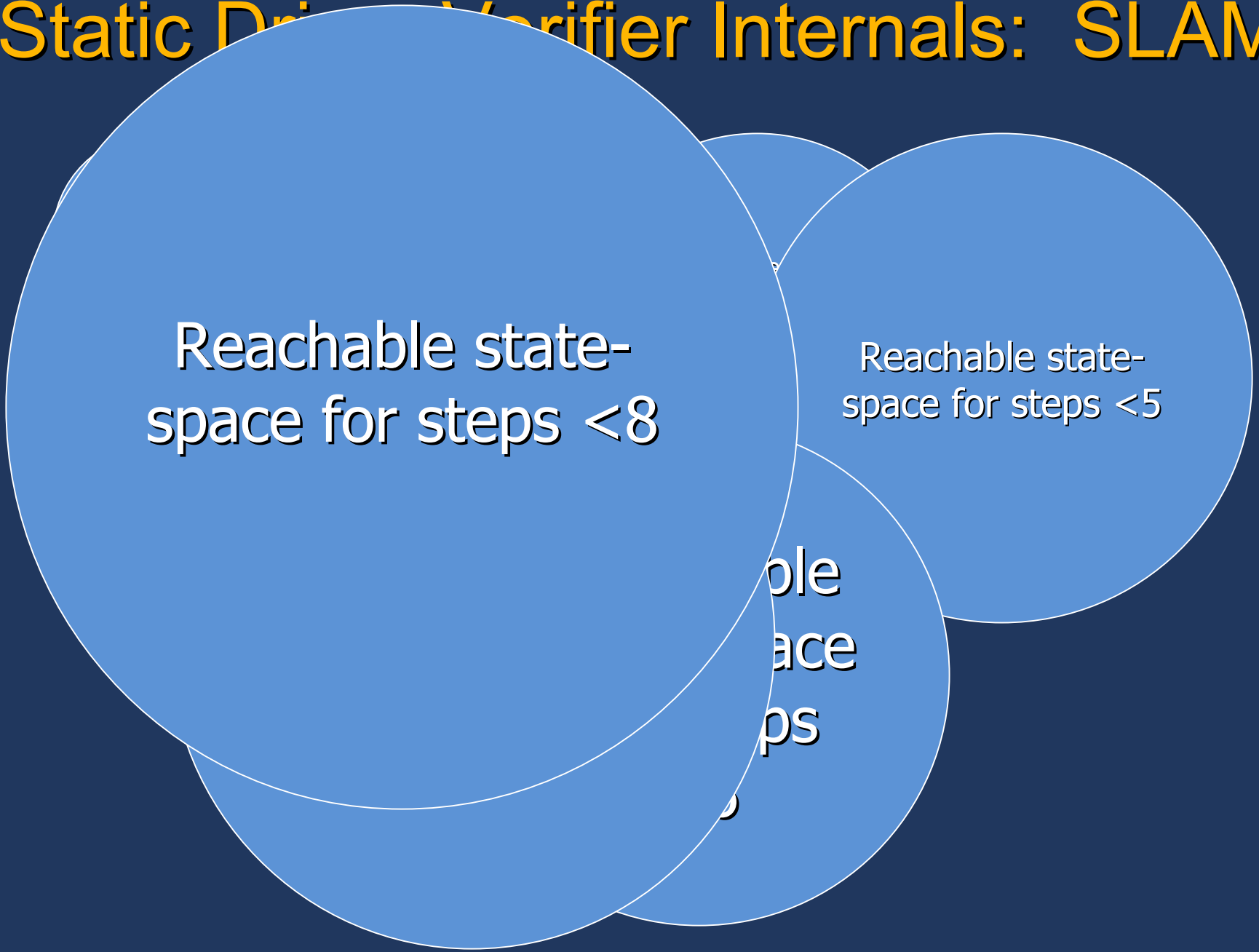
Static Driver Verifier Internals: SLAM



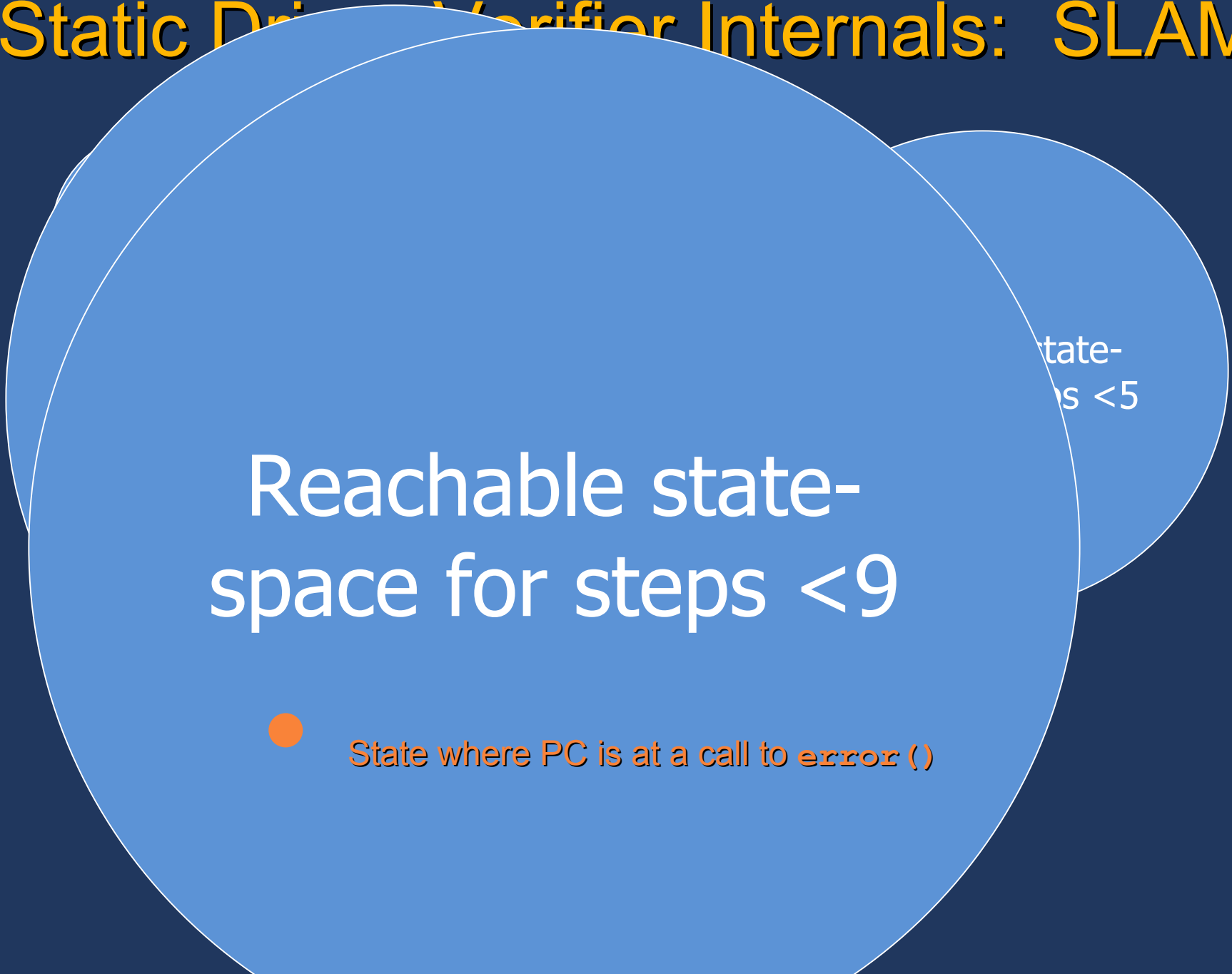
Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM

```
void main()
{


    for(;;) {
        AcquireLock();


        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM

```
void main()
{


    for(;;) {
        AcquireLock();


        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM

```
void main()
{

    for(;;) {
        AcquireLock();

        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM

```
void main()
{

    for(;;) {
        AcquireLock();

        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM

```
void main()
{

    for(;;) {
        AcquireLock();

        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM

```
void main()
{


    for(;;) {
        AcquireLock();


        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM

```
void main()
{


    for(;;) {
        AcquireLock();


        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM

```
void main()
{


    for(;;) {
        AcquireLock();


        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
        }
        g();
    }
}
```

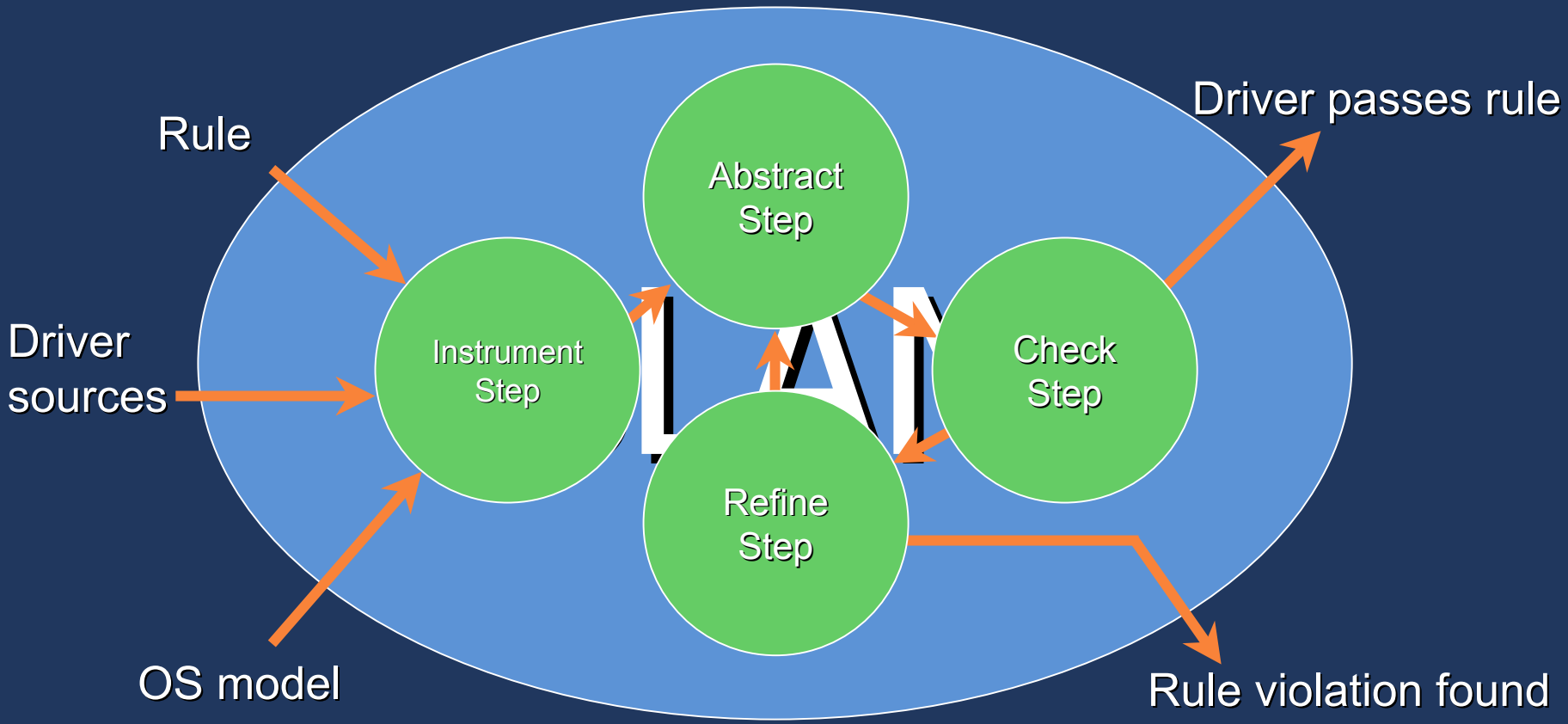
Static Driver Verifier Internals: SLAM

```
void main()
{

    for(;;) {
        AcquireLock();

        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM

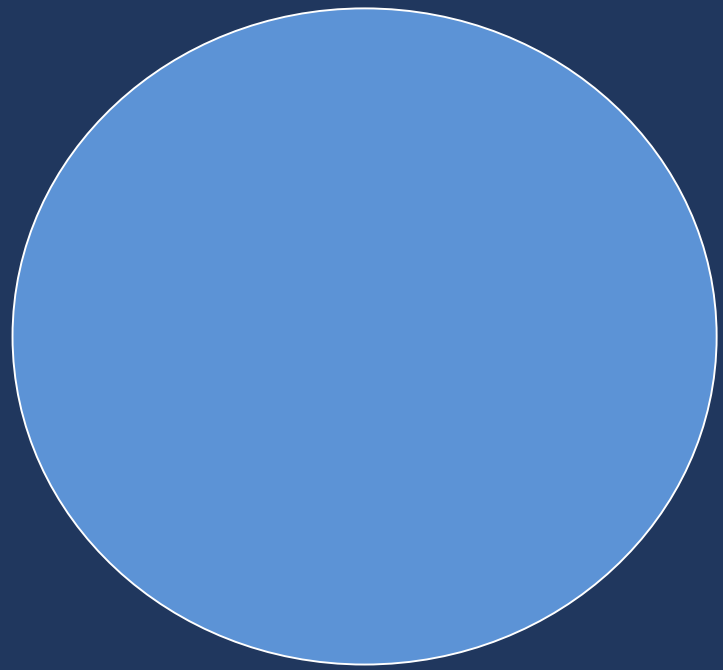
```
void main()
{

    for(;;) {
        AcquireLock();

        while( *  ) {
                f1();
                f2();
                f3();
            if (     *     ) {

                ReleaseLock();
            }
         }
         g();
    }
}
```

Static Driver Verifier Internals: SLAM



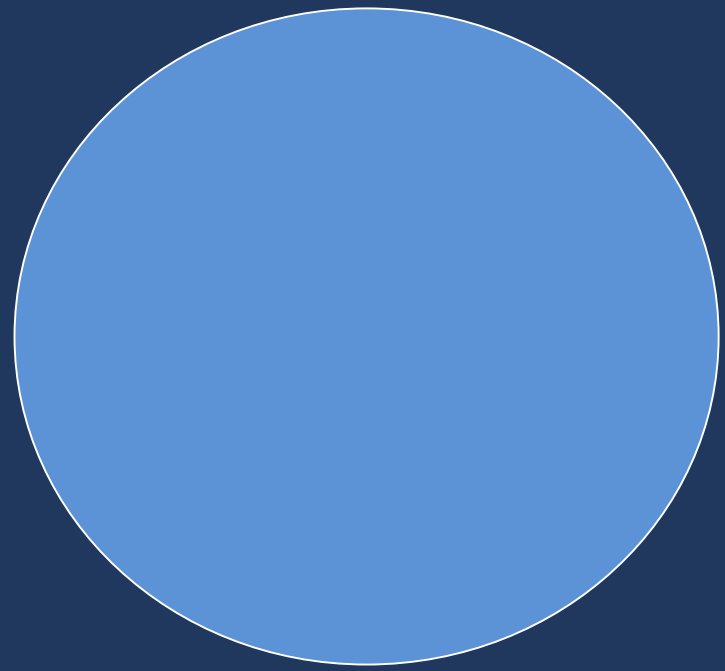
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



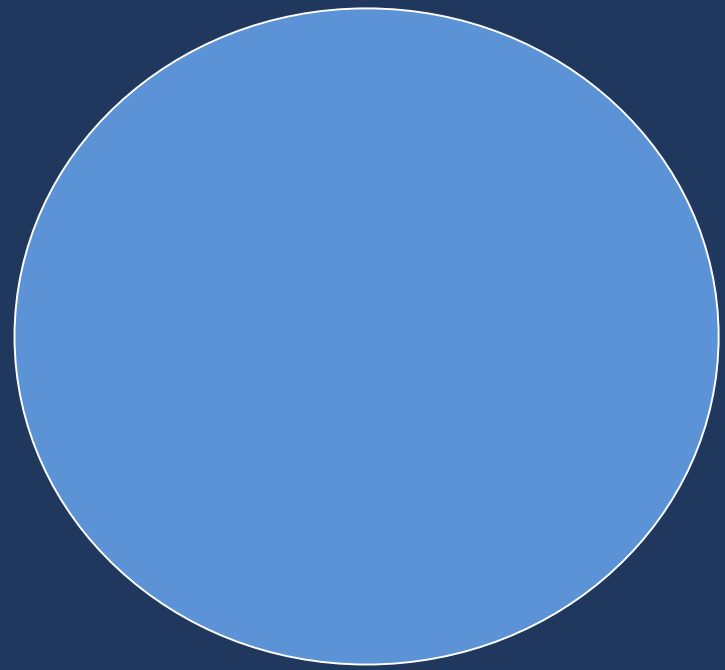
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



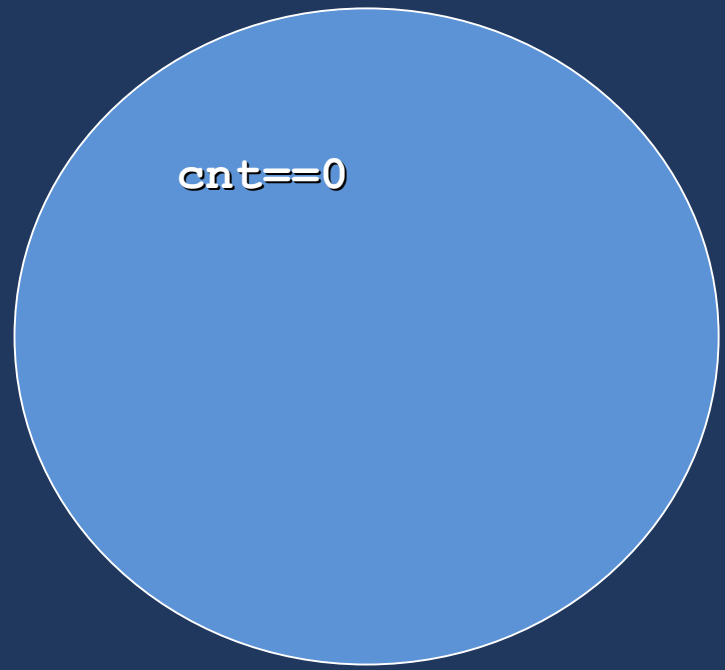
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



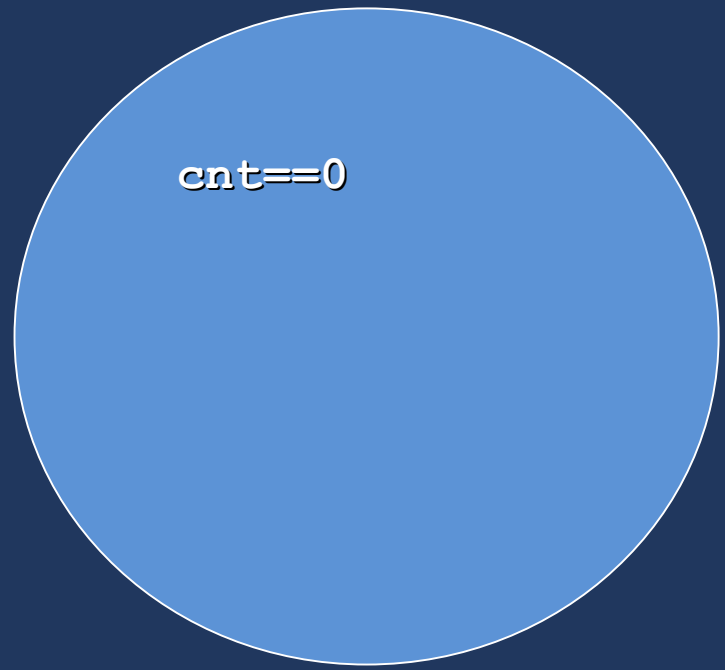
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



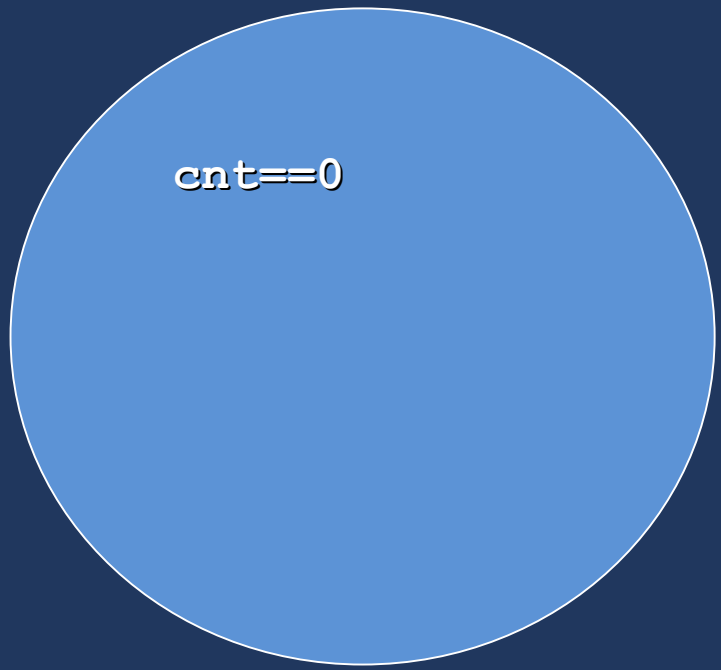
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



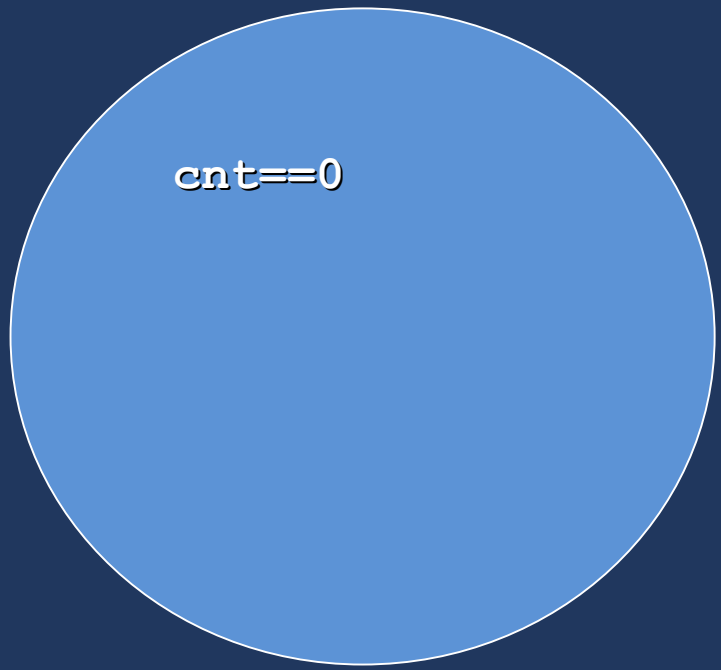
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



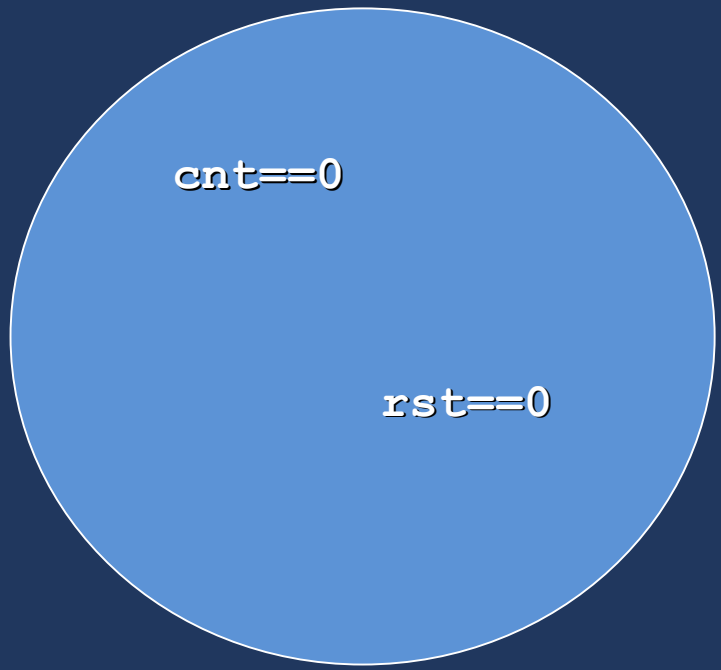
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



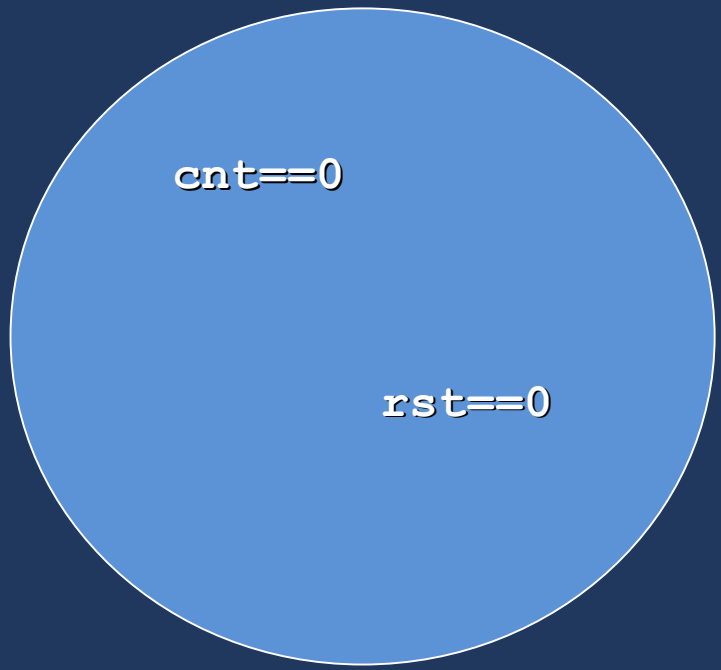
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



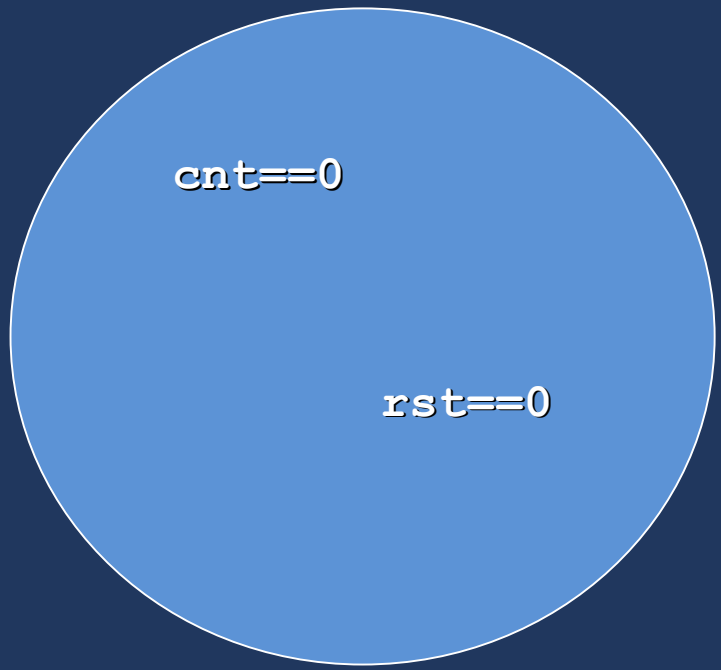
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



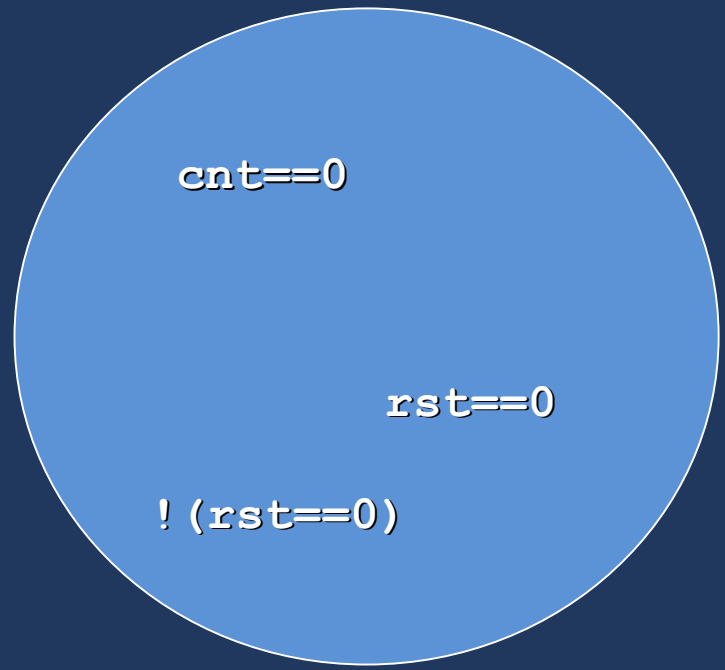
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



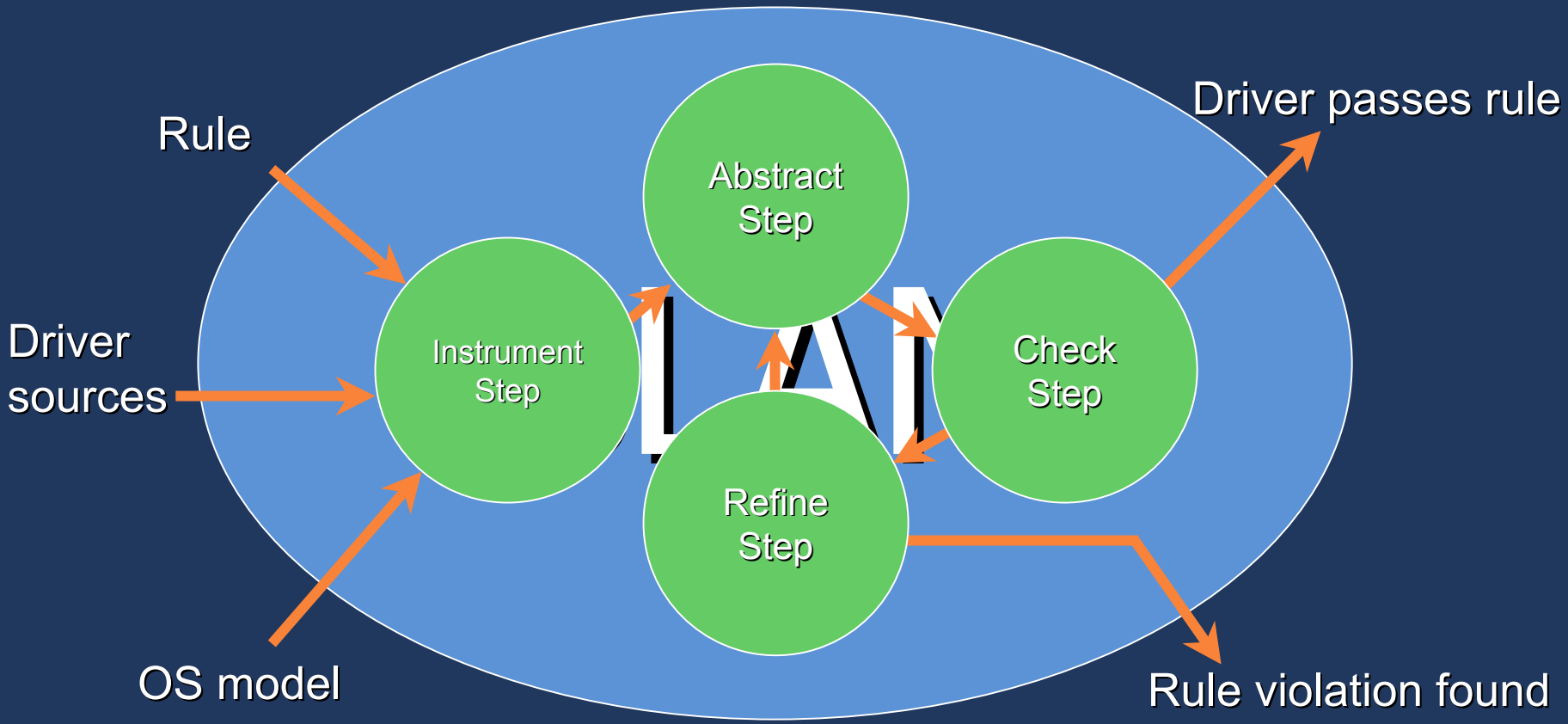
Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    cnt = 0;
    for(;;) {
        AcquireLock();
        rst=0;

        while(!rst) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                rst=1;
                ReleaseLock();
            }
        }
        g();
    }
}
```



New predicate to track: `main { rst==0 }`

Static Driver Verifier Internals: SLAM

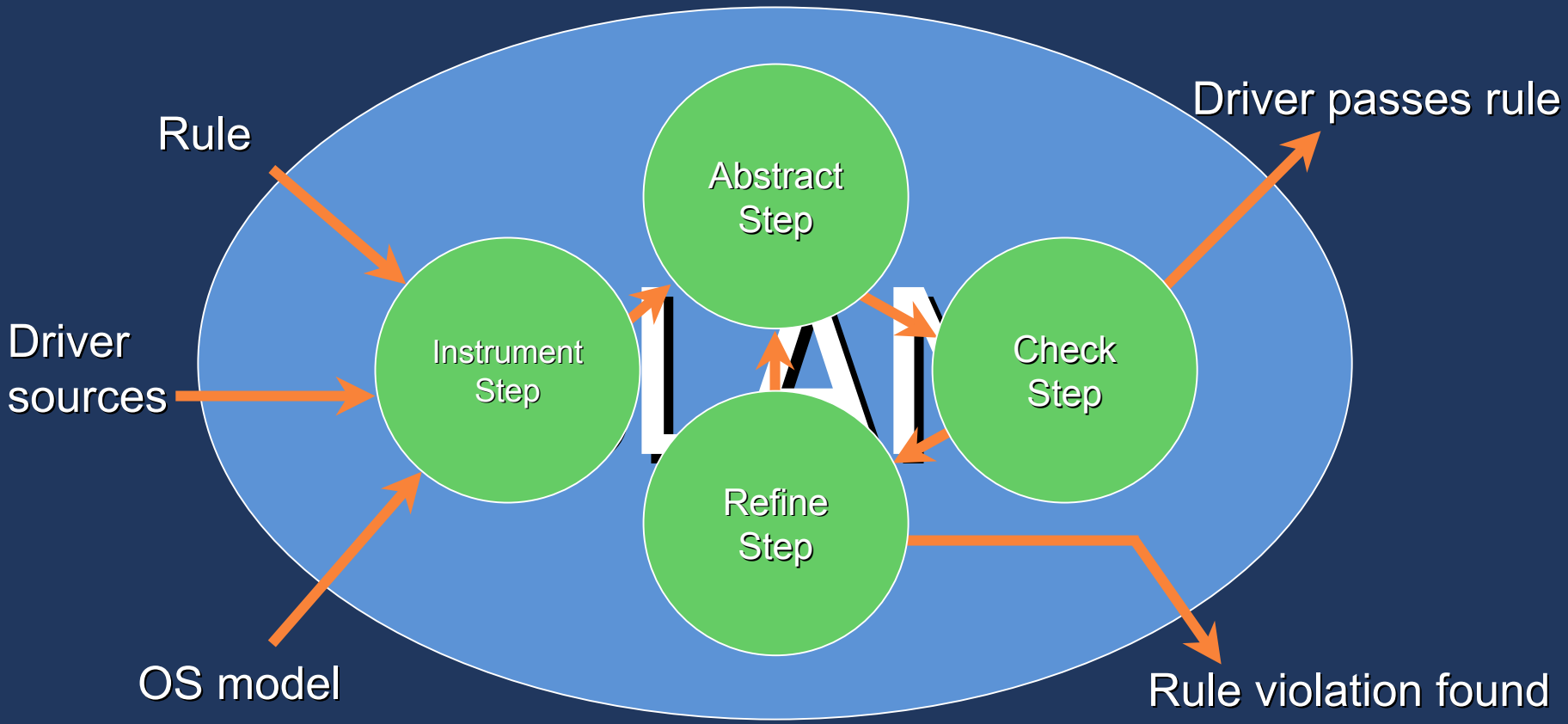


Static Driver Verifier Internals: SLAM

```
void main()
{
    int a,b,c,rst,cnt;
    bool v0; // represents rst==0
    for(;;) {
        AcquireLock();
        v0=1;

        while(!v0 ) {
            a = f1();
            b = f2();
            c = f3();
            if (a<b && b<c) {
                v0=0;
                ReleaseLock();
            }
        }
        g();
    }
}
```

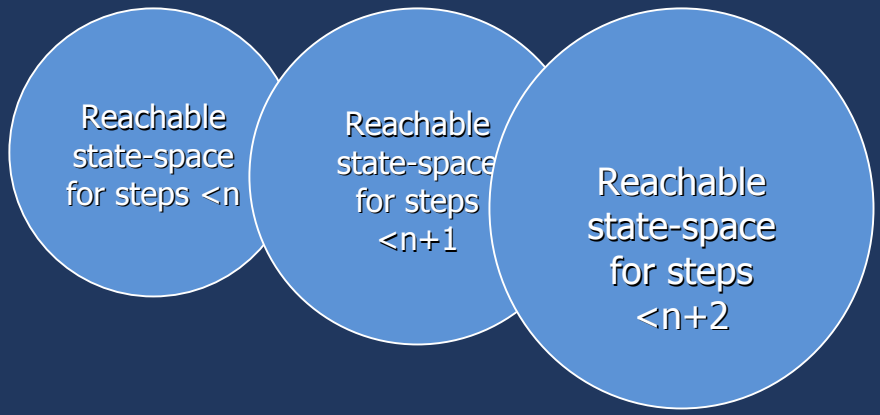
State space = 2^(3 + #bits(pc)) + stack

Static Driver Verifier Internals: SLAM

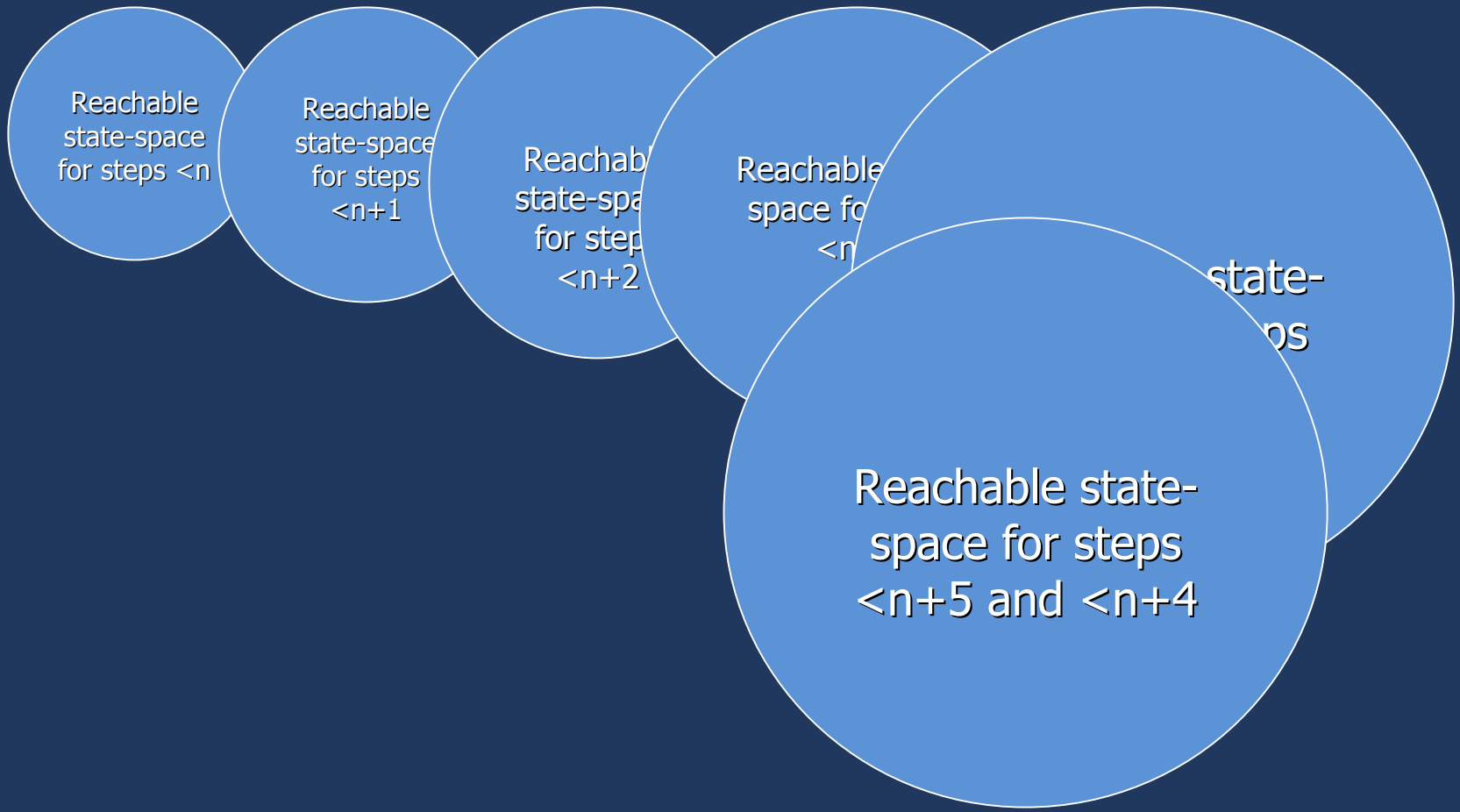


Static Driver Verifier Internals: SLAM

Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM



Static Driver Verifier Internals: SLAM

- The abstraction contains only the PC and these three state bits
  - locked==1
  - locked==0
  - rst==0
- Abstracted away
  - Much of f1(), f2(), f3(), g(),
  - cnt,
  - a, b, c
  - Potential values from rst
- From this abstraction we can reasons that the original C program is also correct

Summary

- SDV is a compile-time tool that finds bugs in device drivers
  - Finds bugs that are harder to reproduce than Driver Verifier does
  - Finds bugs that are deeper than PREfast for Drivers does
- SDV is composed of three parts
  - Rules provided
  - Model of the OS provided
  - SLAM: a symbolic model checker that finds abstractions via refinement
- SDV is being used internally
  - Device driver writers are beginning to use it
  - We've tried it on 3rd party drivers
- SDV is planned to be part of DDK in the future

demo

Resources

- Email
  - Sdvfdbk @ microsoft.com
- White Paper
  - Static Driver Verifier: Finding Bugs in Device Drivers at Compile-Time on http://www.microsoft.com/whdc/devtools/tools/SDV.mspx
- Microsoft Hardware Newsletter
  - Subscribe to the Microsoft Hardware Newsletter and watch the WHDC web site for information about future beta and final releases of new tools from Microsoft

Question & Answer