CS452/652 Real-Time Programming Course Notes

Daniel M. Berry, Cheriton School of Computer Science
University of Waterloo

Three Levels of Task Switching

1. High: Unbounded number of processors
2. Middle: Unbounded number of processes, one CPU, and kernel routines
3. Low: Unbounded number of processes, one CPU, and kernel task

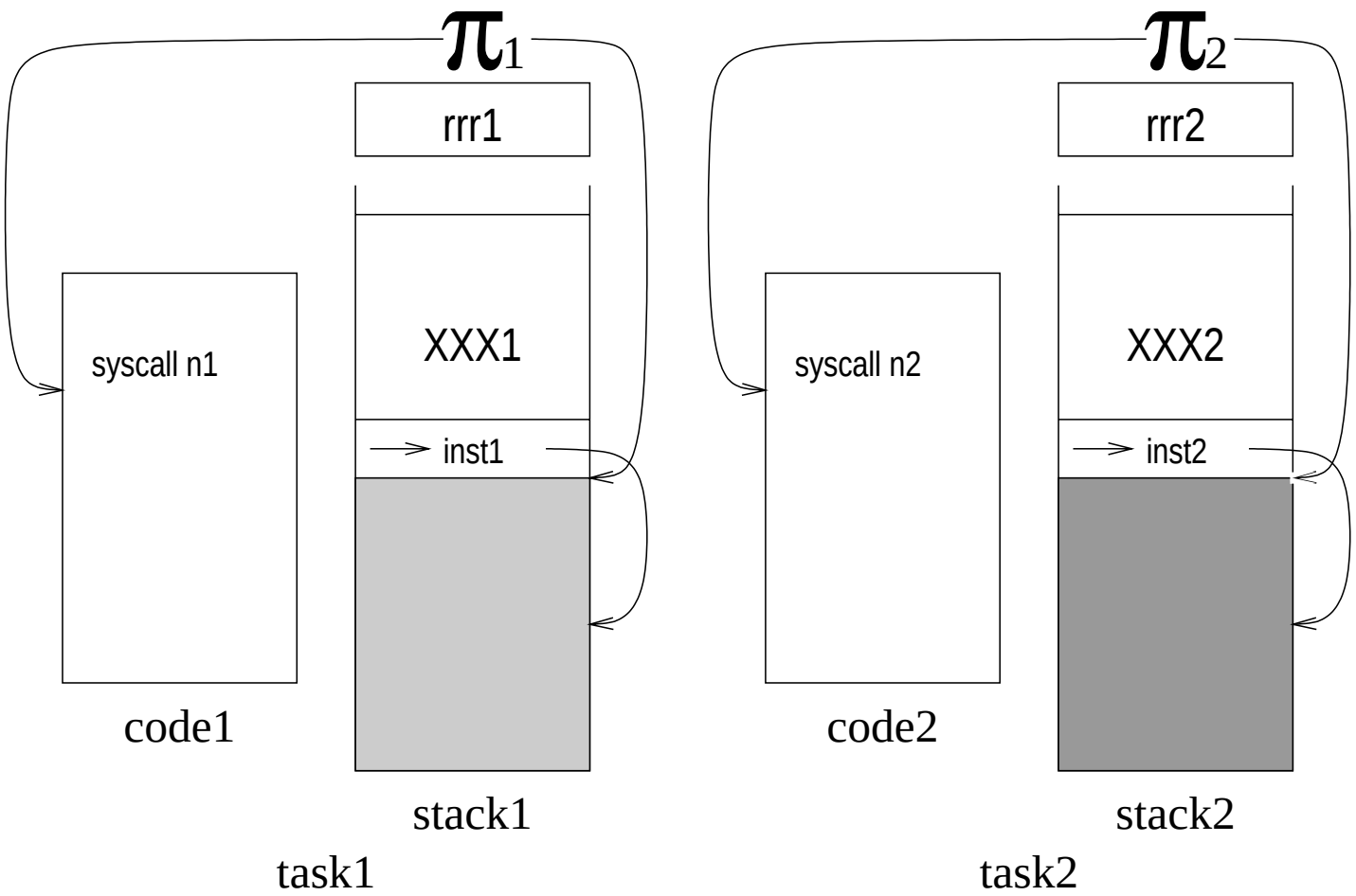
High Level View

Unbounded number of processors:

A system service is either

- instantaneous, like an ordinary instruction, or
- blocking, causing the executing processor to be blocked.

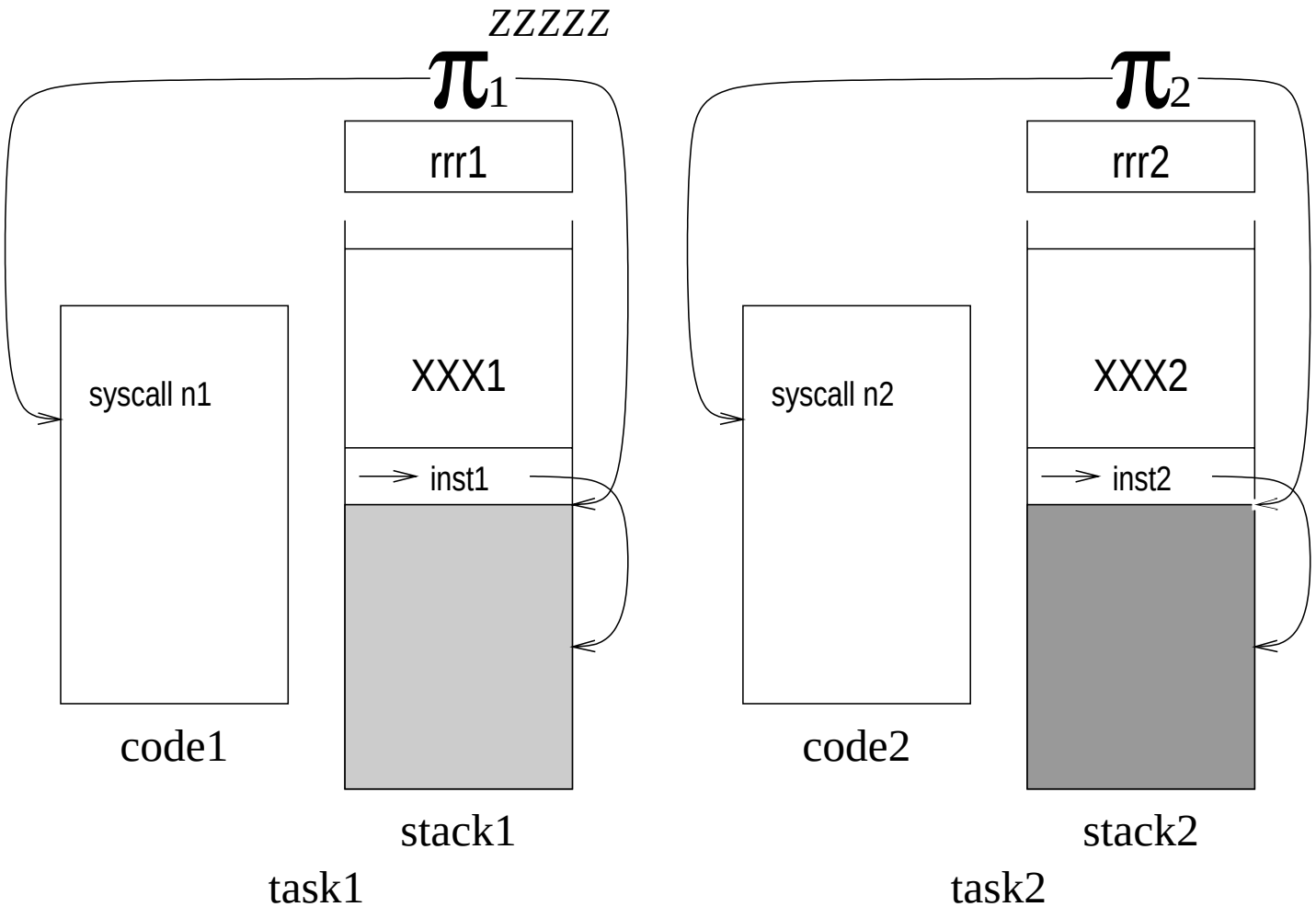
Distinction between running and ready does not exist.

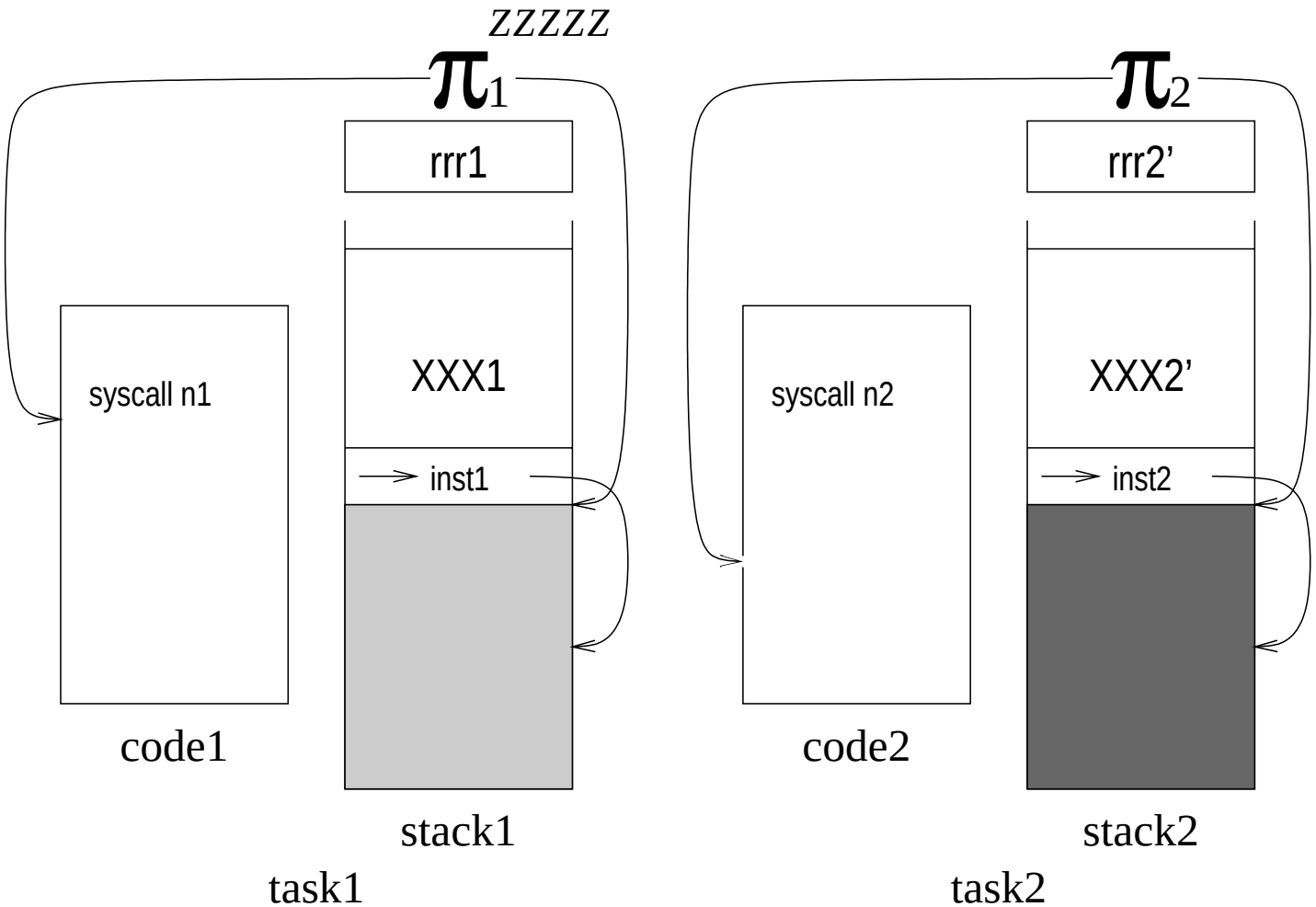

π1
rrr1
XXX1
syscall n1
inst1
code1
stack1
task1
π2
rrr2
XXX2
syscall n2
inst2
code2
stack2
task2

High Level View, Cont'd

In the previous snapshot, both processors are running.

Next two snapshots shows one processor blocked (asleep) and the other running.


zzzzz
π1
rrr1
XXX1
syscall n1
inst1
code1
stack1
task1
π2
rrr2
XXX2
syscall n2
inst2
code2
stack2
task2


zzzzz
π1
rrr1
XXX1
syscall n1
inst1
code1
stack1
task1
π2
rrr2'
XXX2'
syscall n2
inst2
code2
stack2
task2

Middle Level View

Unbounded number of processes, one CPU, and kernel routines

That is, a system service is done in a non-task kernel that is made up of procedures that can be called by the CPU.

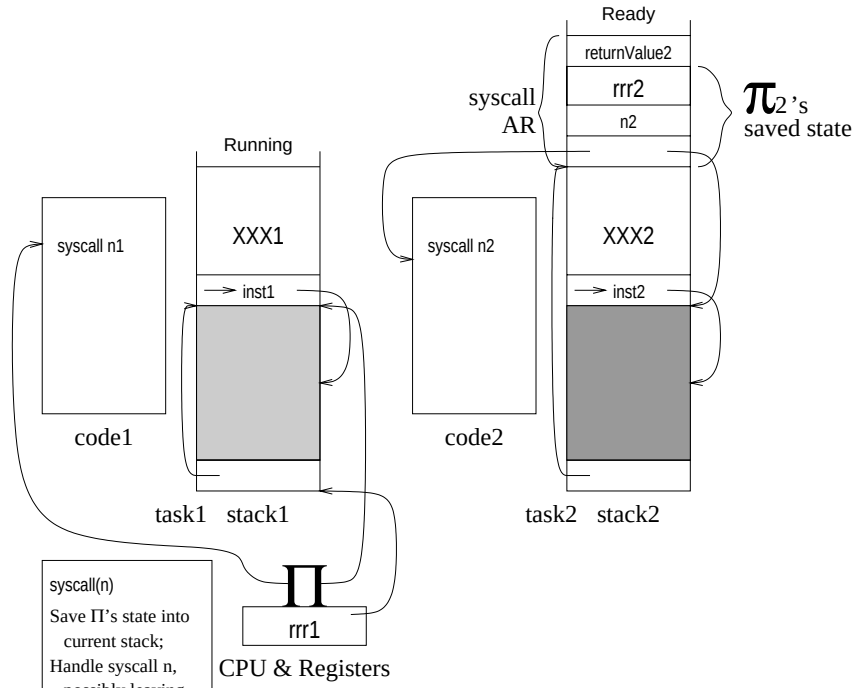
Middle Level View, Cont'd

Task switching is done as a side effect of making a syscall, which does both the requested service and switches to the task with the highest priority, usually leaving the calling task blocked or ready.

It is possible that if the calling task were to be left ready, it might be chosen as the task to switch to, but only if it is the task with the highest priority.

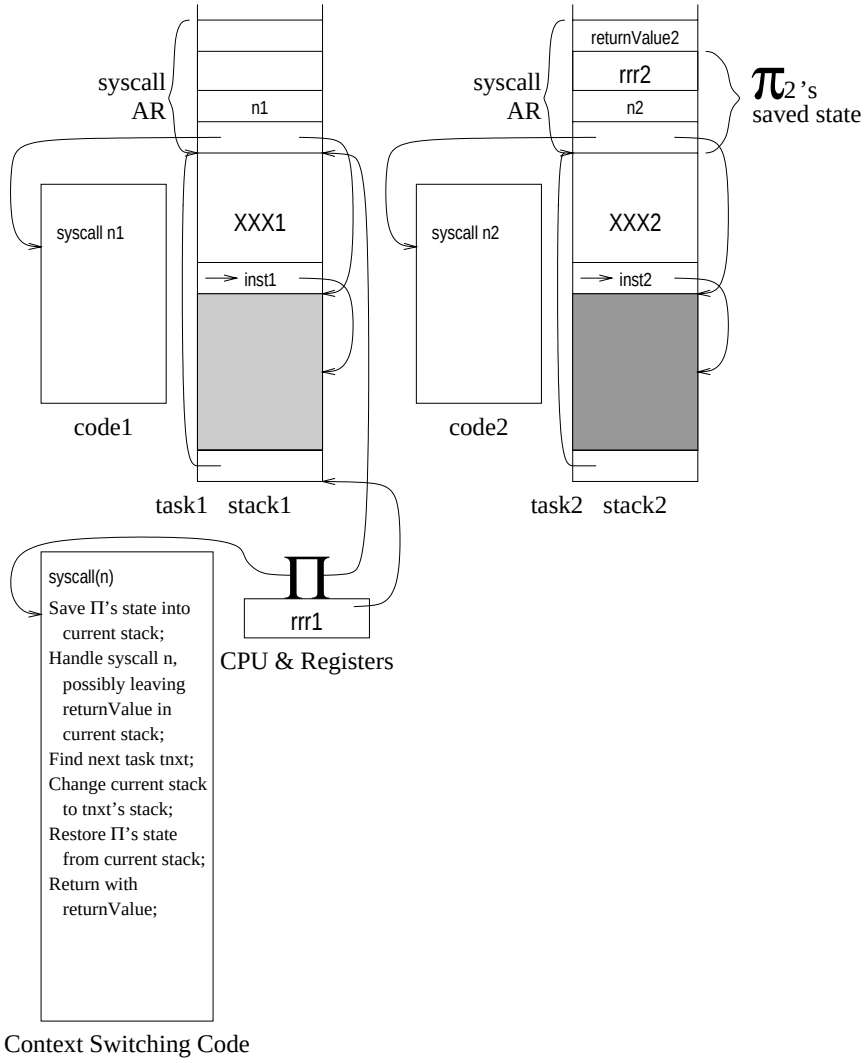
Middle Level View, Cont'd

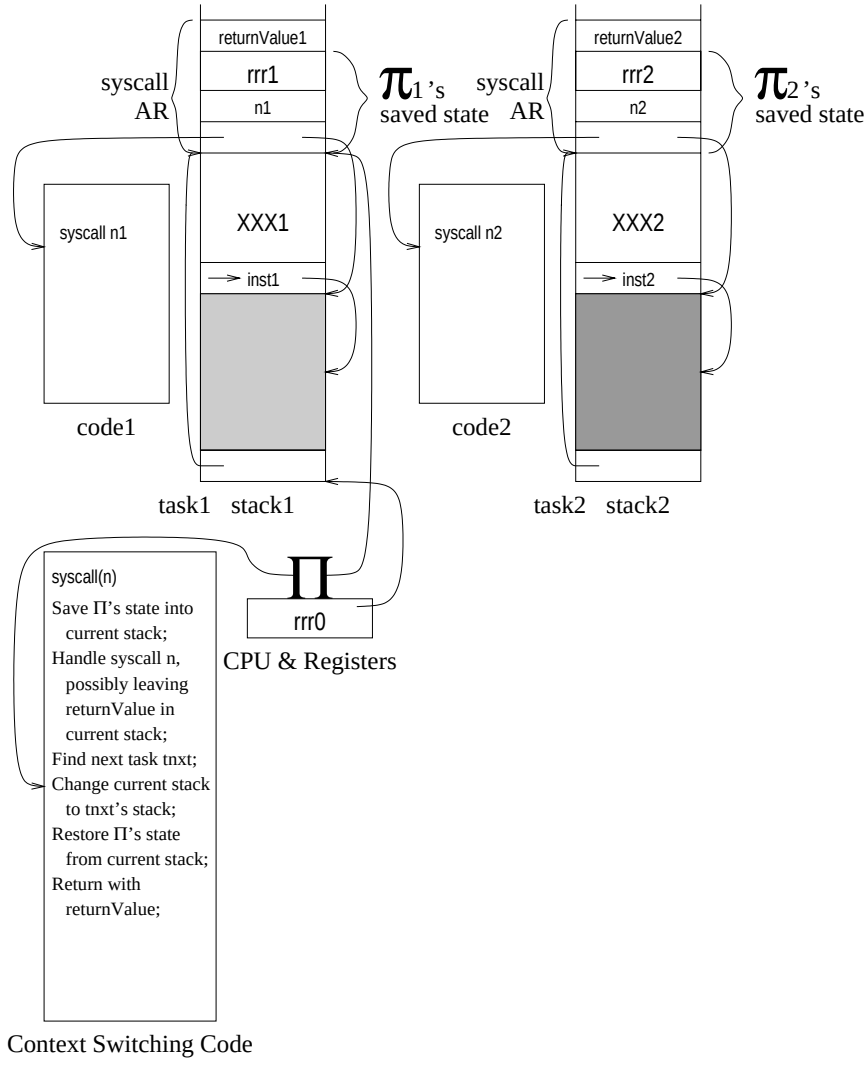
Note that any task that is not running is left blocked or ready in the middle of the syscall routine. So task switching is done by changing the stack and base pointers while keeping the instruction pointer pointing into the syscall routine.




```
syscall(n)
Save Π's state into
   current stack;
Handle syscall n,
   possibly leaving
   returnValue in
   current stack;
Find next task tnxt;
Change current stack
   to tnxt's stack;
Restore Π's state
   from current stack;
Return with
   returnValue;
```

Context Switching Code


syscall
AR
n1
XXX1
inst1
syscall n1
code1
task1 stack1
returnValue2
rrr2
n2
XXX2
inst2
syscall n2
code2
task2 stack2
π2's
saved state
Π
rrr1
CPU & Registers
syscall(n)
Save Π's state into
current stack;
Handle syscall n,
possibly leaving
returnValue in
current stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
Return with
returnValue;
Context Switching Code


returnValue1
rrr1
n1
syscall
AR
π1's
saved state
returnValue2
rrr2
n2
syscall
AR
π2's
saved state
syscall n1
XXX1
inst1
code1
task1 stack1
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrr0
CPU & Registers
syscall(n)
Save Π's state into
current stack;
Handle syscall n,
possibly leaving
returnValue in
current stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
Return with
returnValue;
Context Switching Code

“possibly leaving returnValue”?

Why does the instruction say “Handle syscall n, *possibly leaving* returnValue in current stack;” instead of just “leaving”?

“possibly …”, Cont’d

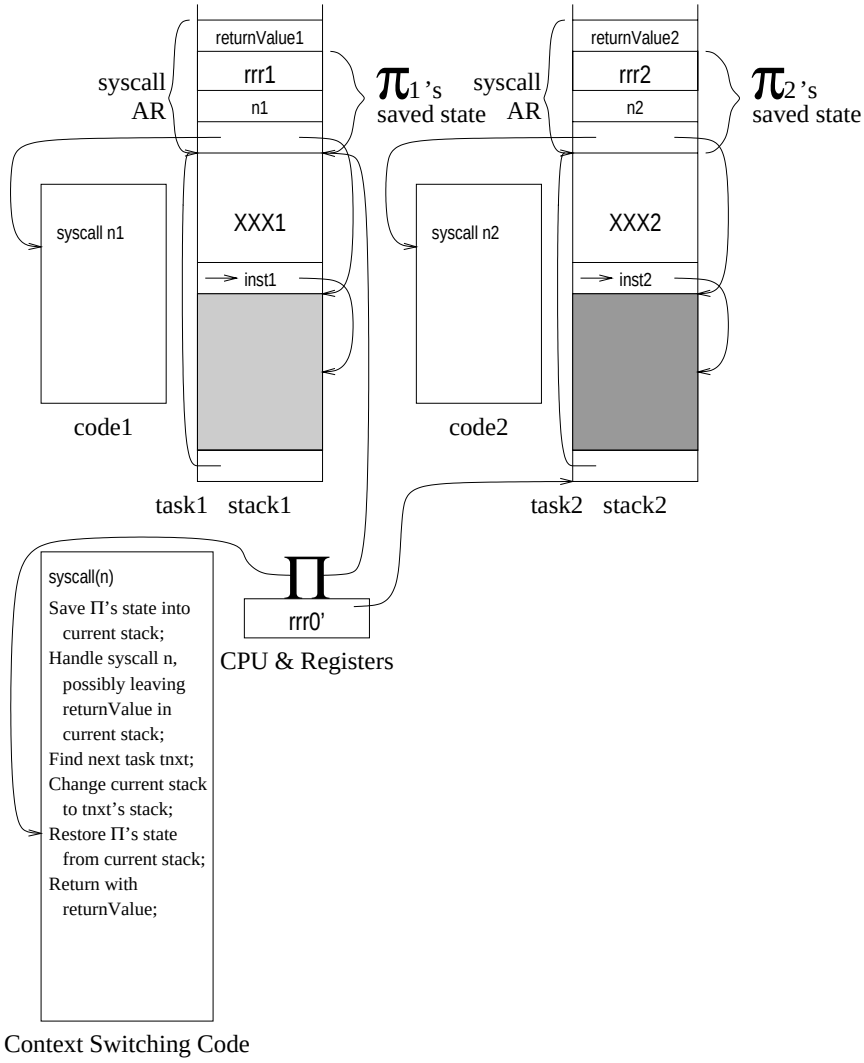
If the syscall is an instantaneous kind, then handling syscall n will definitely leave returnValue in current stack.

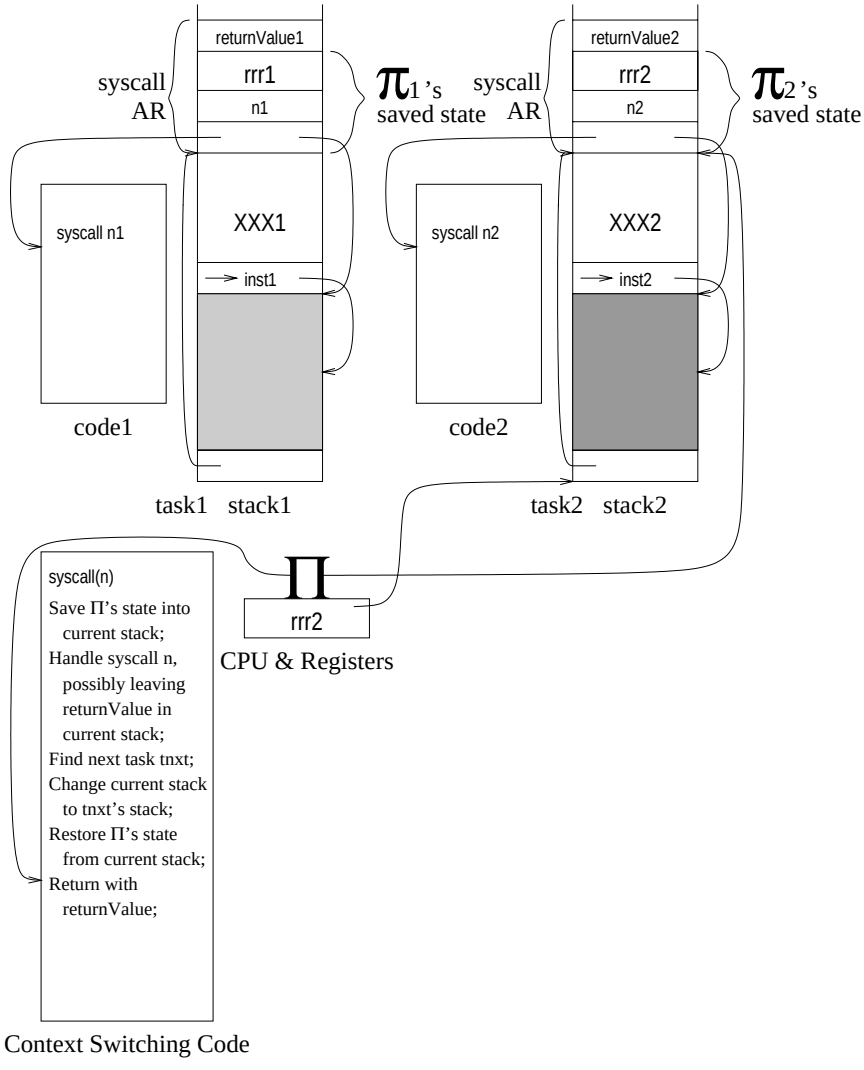
“possibly …”, Cont’d

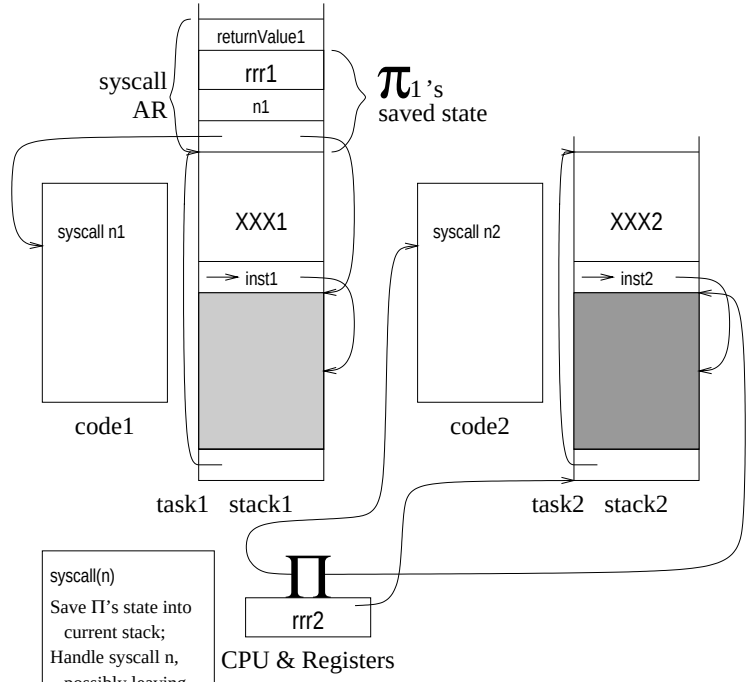
If, on the other hand, the syscall is a blocking kind, then handling syscall n results in arranging that some later execution of syscall will discover that the data requested in this syscall are ready, that its value should be deposited at the top of *this stack*, via a pointer associated with the expected value, and that *this stack* should be changed from blocked to ready.

“possibly …”, Cont’d

The difficulty of keeping track of these pending completions of syscalls through multiple invocations of syscalls by one CPU, each with its own activation record, is the reason that it is nice to make the kernel a task that keeps its own data.


returnValue1
rrr1
n1
syscall
AR
π1's
saved state
returnValue2
rrr2
n2
syscall
AR
π2's
saved state
syscall n1
XXX1
inst1
code1
task1 stack1
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrr0'
CPU & Registers
syscall(n)
Save Π's state into
current stack;
Handle syscall n,
possibly leaving
returnValue in
current stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
Return with
returnValue;
Context Switching Code


returnValue1
rrr1
n1
syscall
AR
π1's
saved state
returnValue2
rrr2
n2
syscall
AR
π2's
saved state
syscall n1
XXX1
inst1
code1
task1 stack1
syscall n2
XXX2
inst2
code2
task2 stack2
syscall(n)
Save Π's state into
current stack;
Handle syscall n,
possibly leaving
returnValue in
current stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
Return with
returnValue;
Π
rrr2
CPU & Registers
Context Switching Code




```
syscall(n)
Save Π's state into
  current stack;
Handle syscall n,
  possibly leaving
  returnValue in
  current stack;
Find next task tnxt;
Change current stack
  to tnxt's stack;
Restore Π's state
  from current stack;
Return with
  returnValue;
```

Context Switching Code

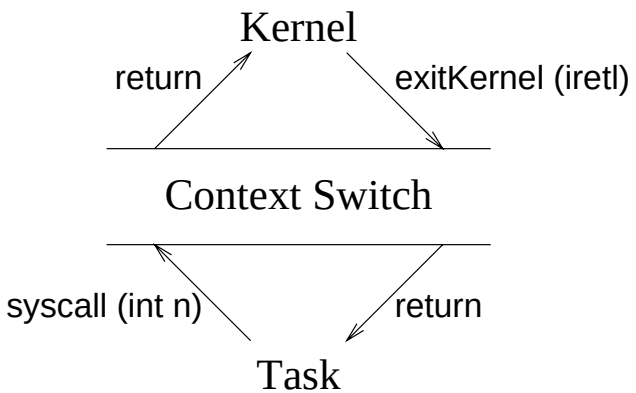
Low Level View

Unbounded number of processes, one CPU, and kernel task

That is, a system service is done in a kernel task.

Low Level View, Cont'd

I tried to decompose the context switching code into the three pieces suggested by the explanation given of the diagram:



for the transition from task1 to the kernel and from the kernel to task2, but it did not quite work.

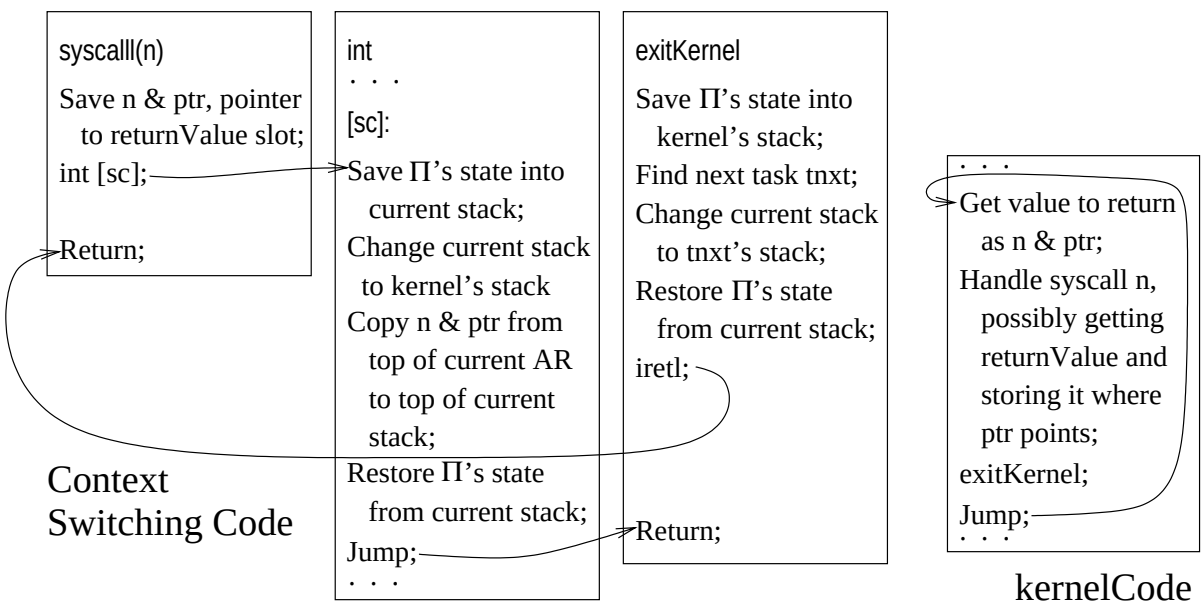
Low Level View, Cont'd

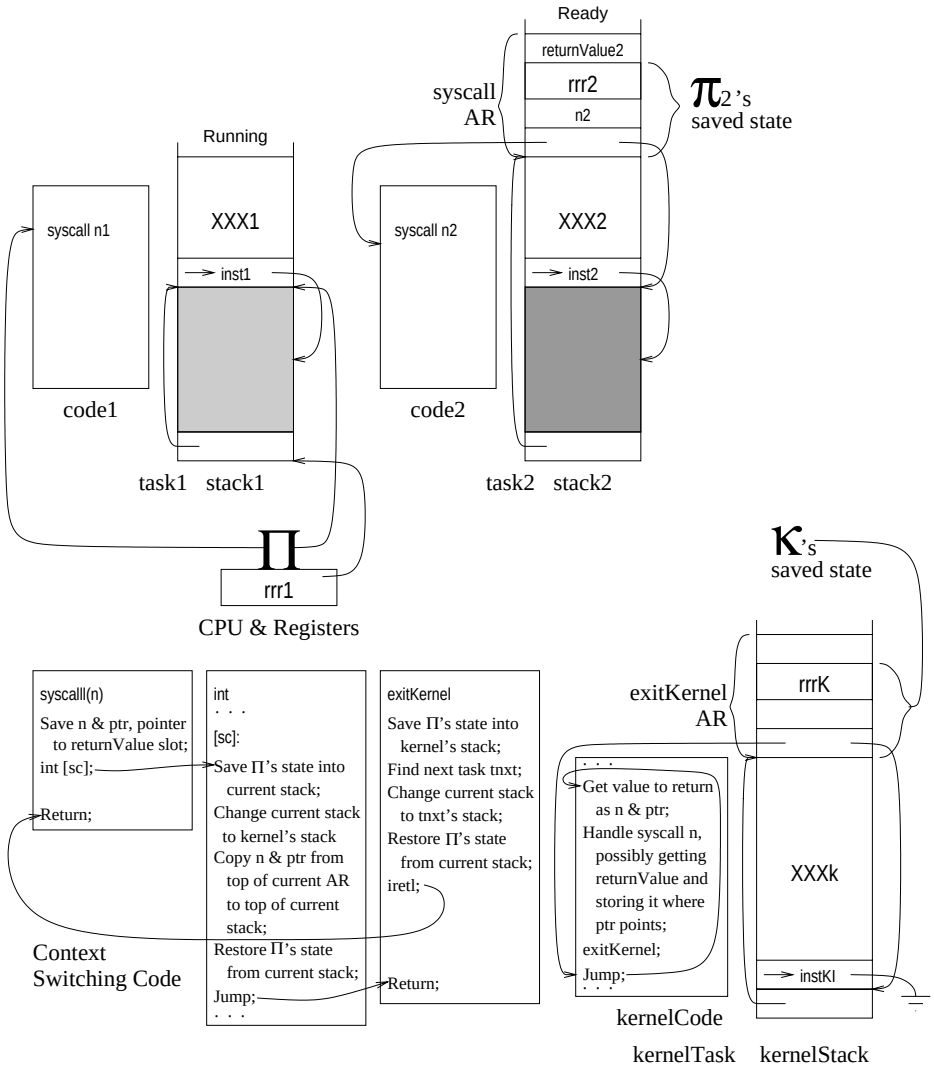
Note that what I show is only *one* possible decomposition of the behavior.

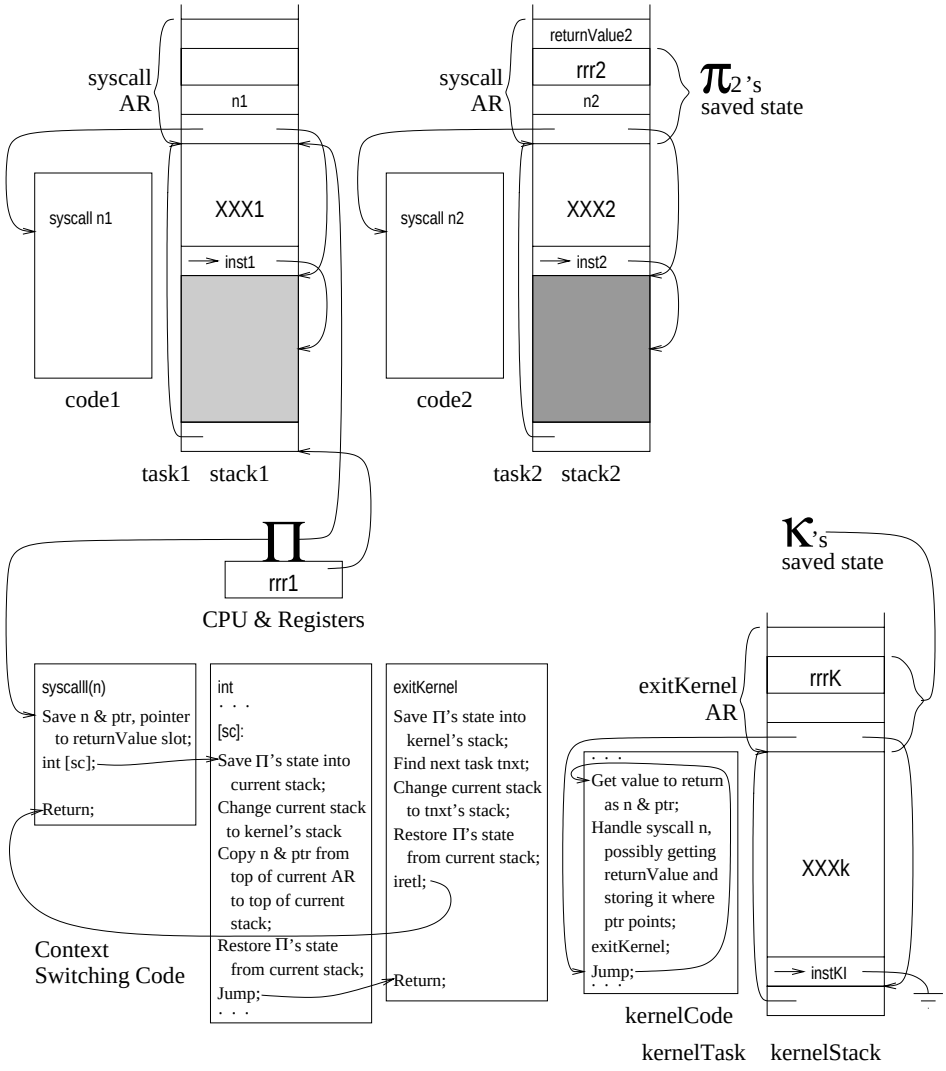
Others will work, in particular e.g., making only one procedure instead of three.

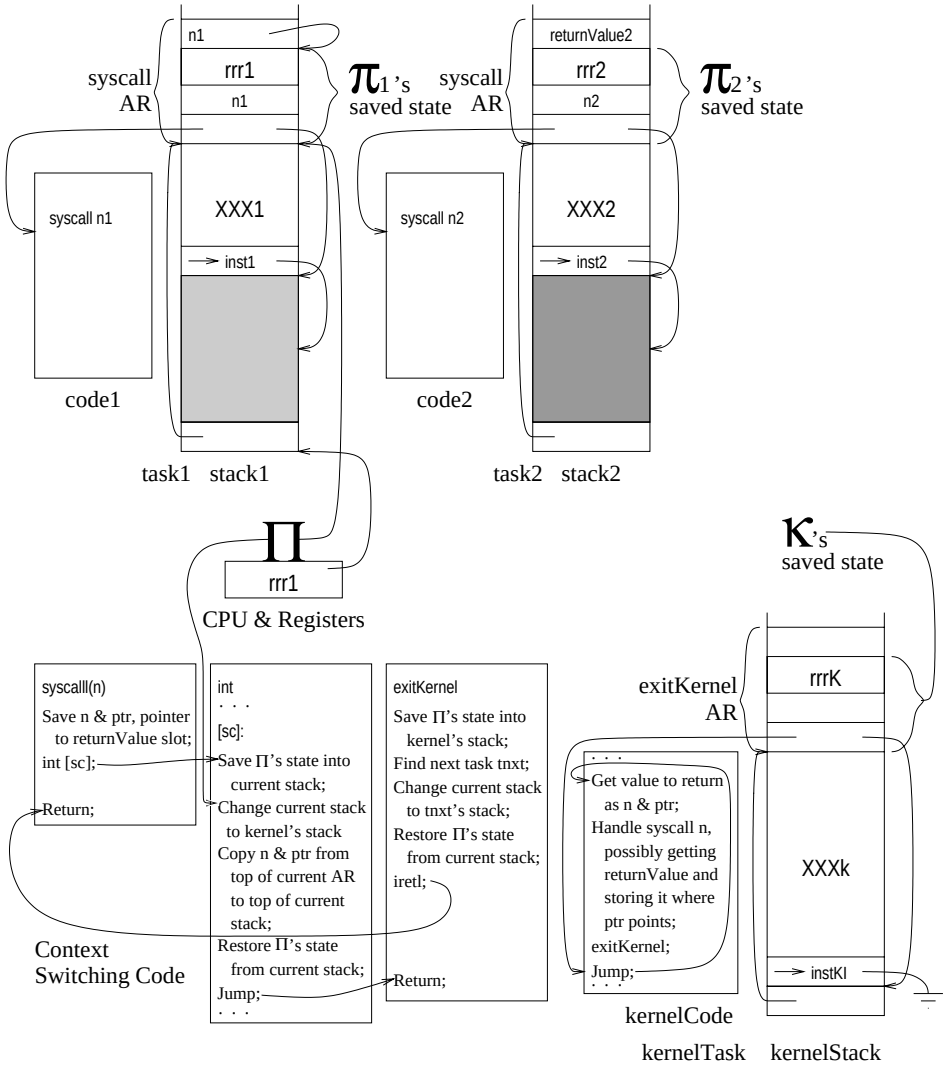
Low Level View, Cont'd

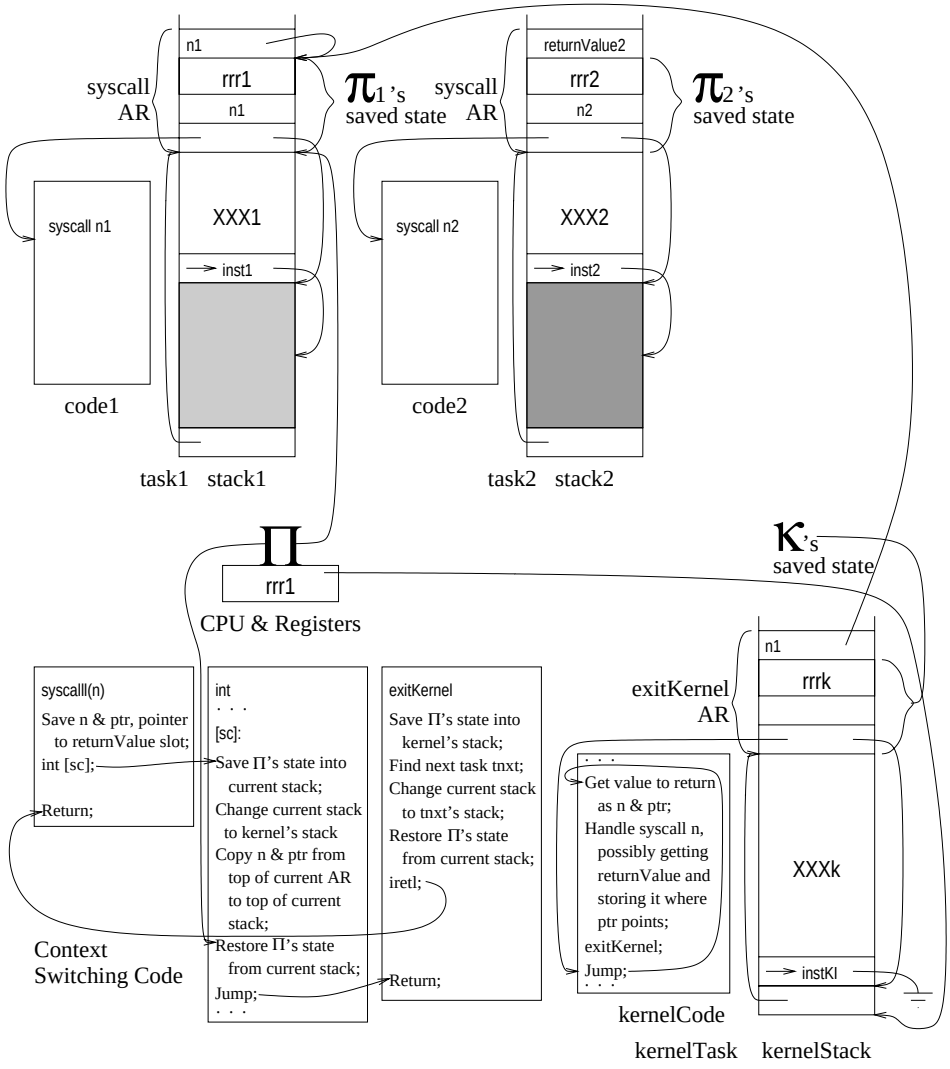
First, a view of the context switching and the kernel code.

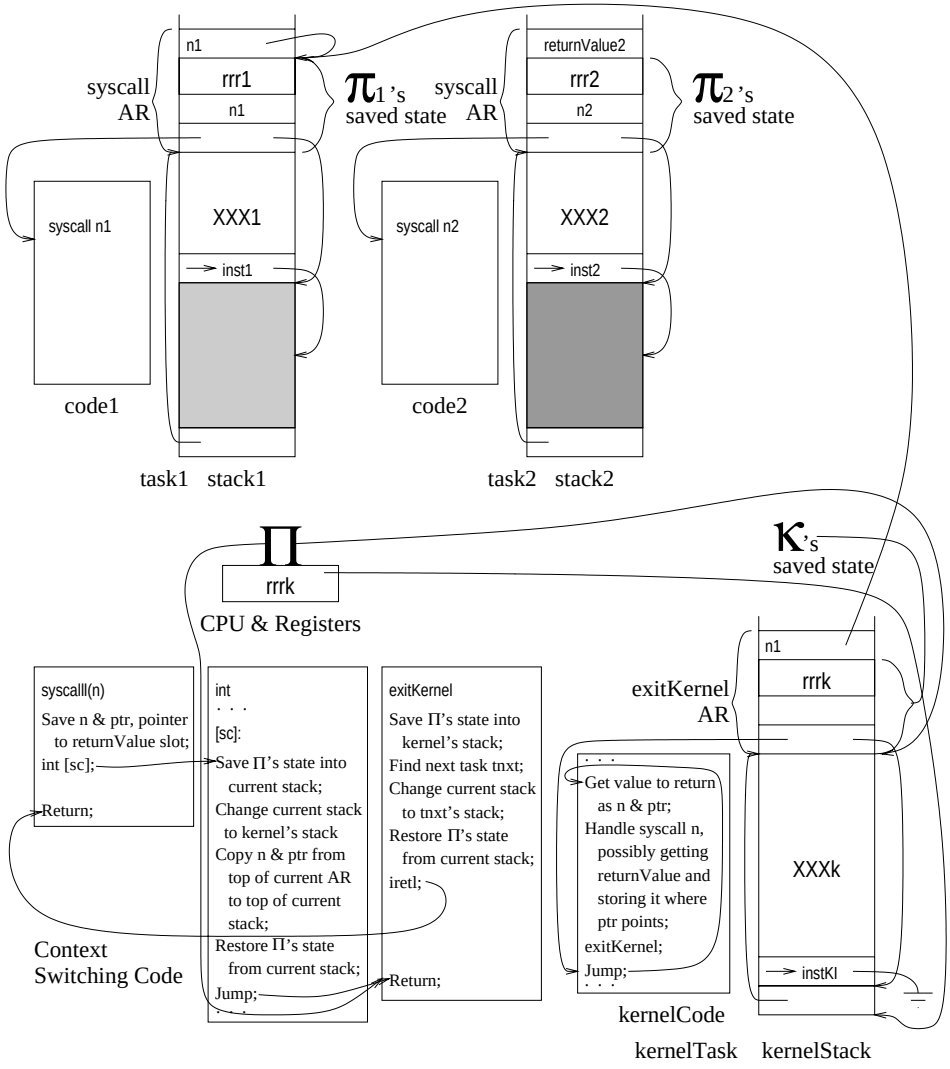


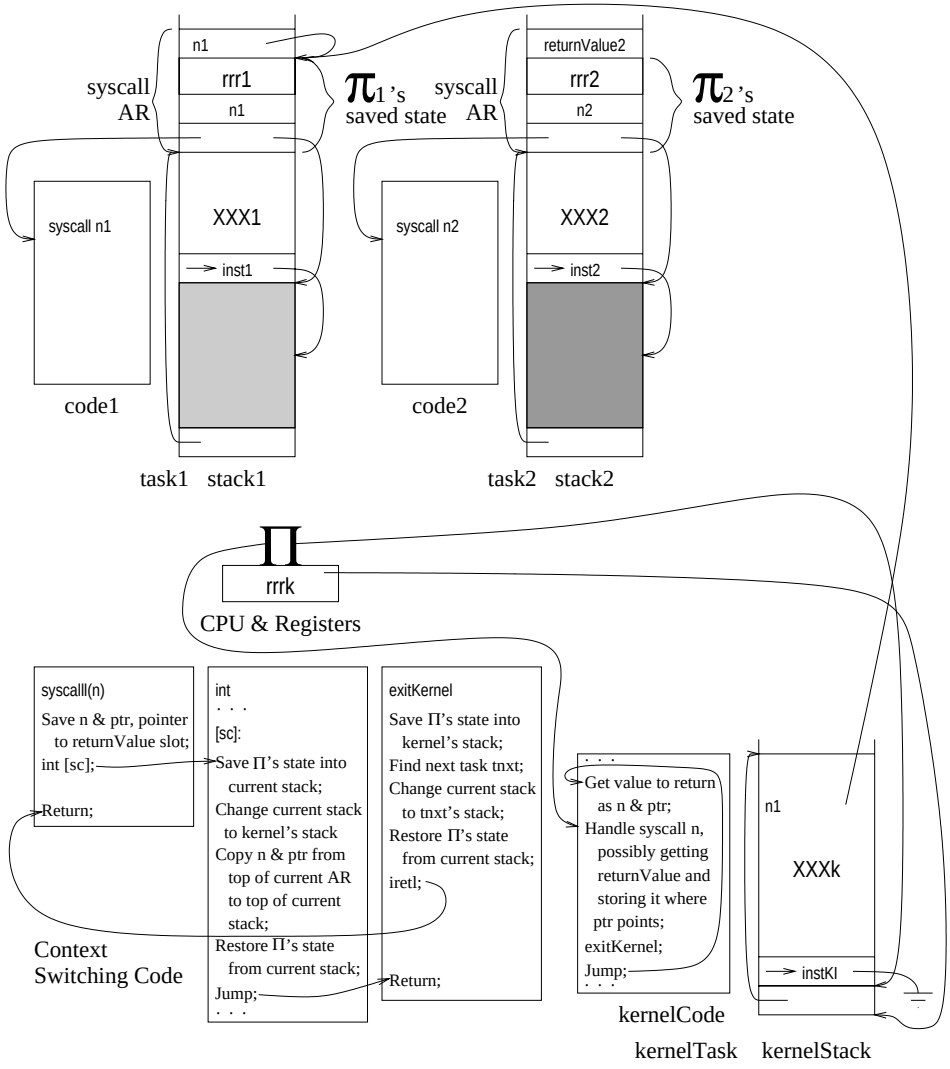

Ready
returnValue2
rrr2
n2
syscall
AR
π2 's
saved state
Running
XXX1
XXX2
syscall n1
syscall n2
inst1
inst2
code1
code2
task1 stack1
task2 stack2
Π
rrr1
CPU & Registers
K's
saved state
rrrK
exitKernel
AR
XXXk
instKI
syscalll(n)
Save n & ptr, pointer
to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into
current stack;
Change current stack
to kernel's stack
Copy n & ptr from
top of current AR
to top of current
stack;
Restore Π's state
from current stack;
Jump;
exitKernel
Save Π's state into
kernel's stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
iretl;
Return;
Get value to return
as n & ptr;
Handle syscall n,
possibly getting
returnValue and
storing it where
ptr points;
exitKernel;
Jump;
Context
Switching Code
kernelCode
kernelTask
kernelStack

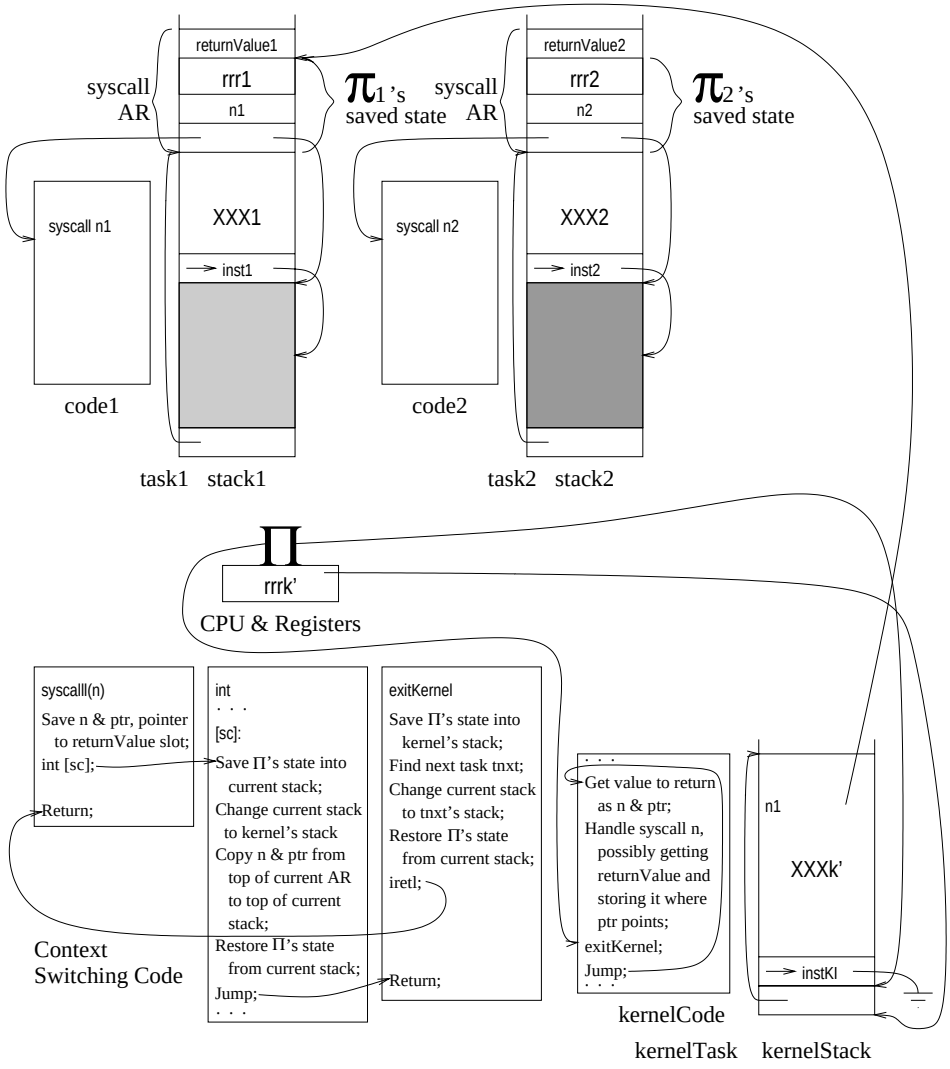

syscall
AR
n1
syscall n1
XXX1
inst1
code1
task1 stack1
returnValue2
rrr2
n2
π2's
saved state
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrr1
CPU & Registers
κ's
saved state
exitKernel
AR
rrrK
XXXk
instKI
syscalll(n)
Save n & ptr, pointer
to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into
current stack;
Change current stack
to kernel's stack
Copy n & ptr from
top of current AR
to top of current
stack;
Restore Π's state
from current stack;
Jump;
exitKernel
Save Π's state into
kernel's stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
iretI;
Return;
Context
Switching Code
Get value to return
as n & ptr;
Handle syscall n,
possibly getting
returnValue and
storing it where
ptr points;
exitKernel;
Jump;
kernelCode
kernelTask
kernelStack

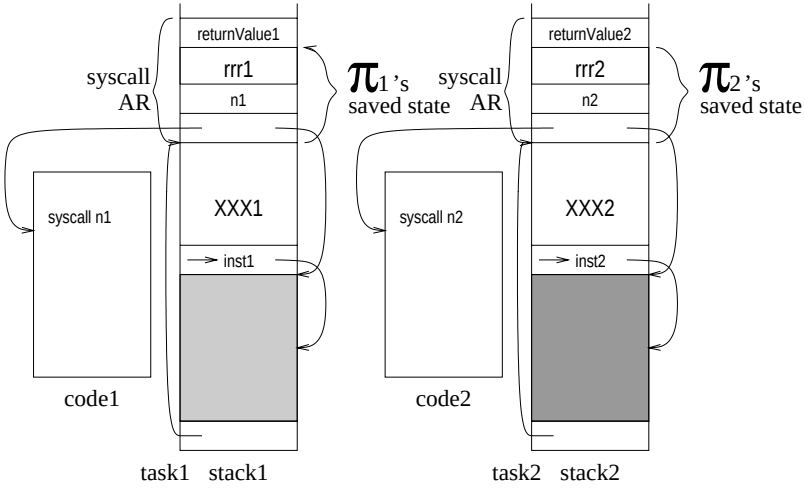

n1
rrr1
n1
syscall AR
π1's saved state
syscall n1
XXX1
inst1
code1
task1 stack1
returnValue2
rrr2
n2
syscall AR
π2's saved state
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrr1
CPU & Registers
κ's saved state
syscalll(n)
Save n & ptr, pointer to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into current stack;
Change current stack to kernel's stack
Copy n & ptr from top of current AR to top of current stack;
Restore Π's state from current stack;
Jump;
exitKernel
Save Π's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore Π's state from current stack;
iretI;
Return;
Context Switching Code
exitKernel AR
rrrK
Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;
XXXk
instKI
kernelCode
kernelTask
kernelStack

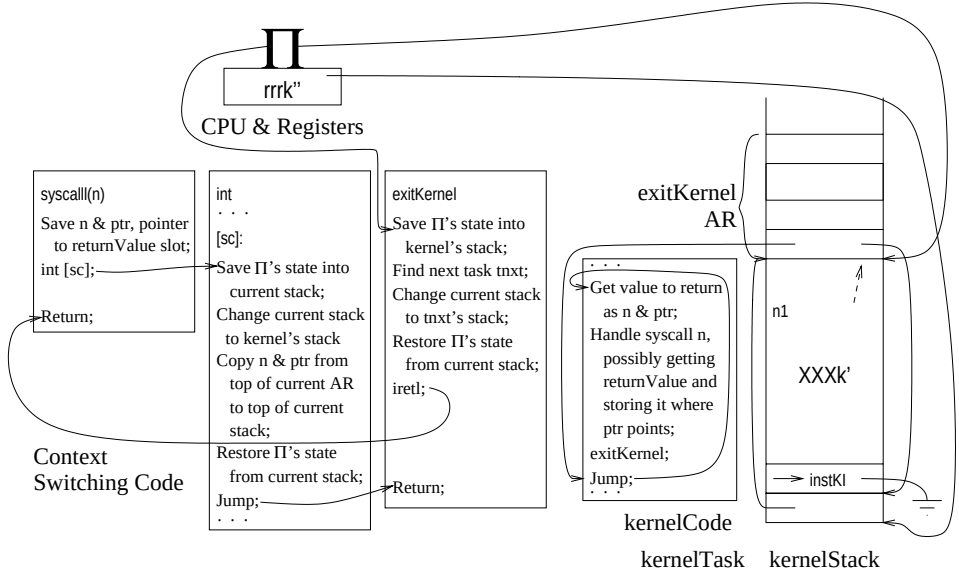

n1
rrr1
n1
syscall
AR
π1's
saved state
returnValue2
rrr2
n2
syscall
AR
π2's
saved state
syscall n1
XXX1
inst1
code1
task1 stack1
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrr1
CPU & Registers
κ's
saved state
n1
rrrk
exitKernel
AR
XXXk
instKl
syscalll(n)
Save n & ptr, pointer
to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into
current stack;
Change current stack
to kernel's stack
Copy n & ptr from
top of current AR
to top of current
stack;
Restore Π's state
from current stack;
Jump;
exitKernel
Save Π's state into
kernel's stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
iretl;
Return;
Get value to return
as n & ptr;
Handle syscall n,
possibly getting
returnValue and
storing it where
ptr points;
exitKernel;
Jump;
Context
Switching Code
kernelCode
kernelTask
kernelStack

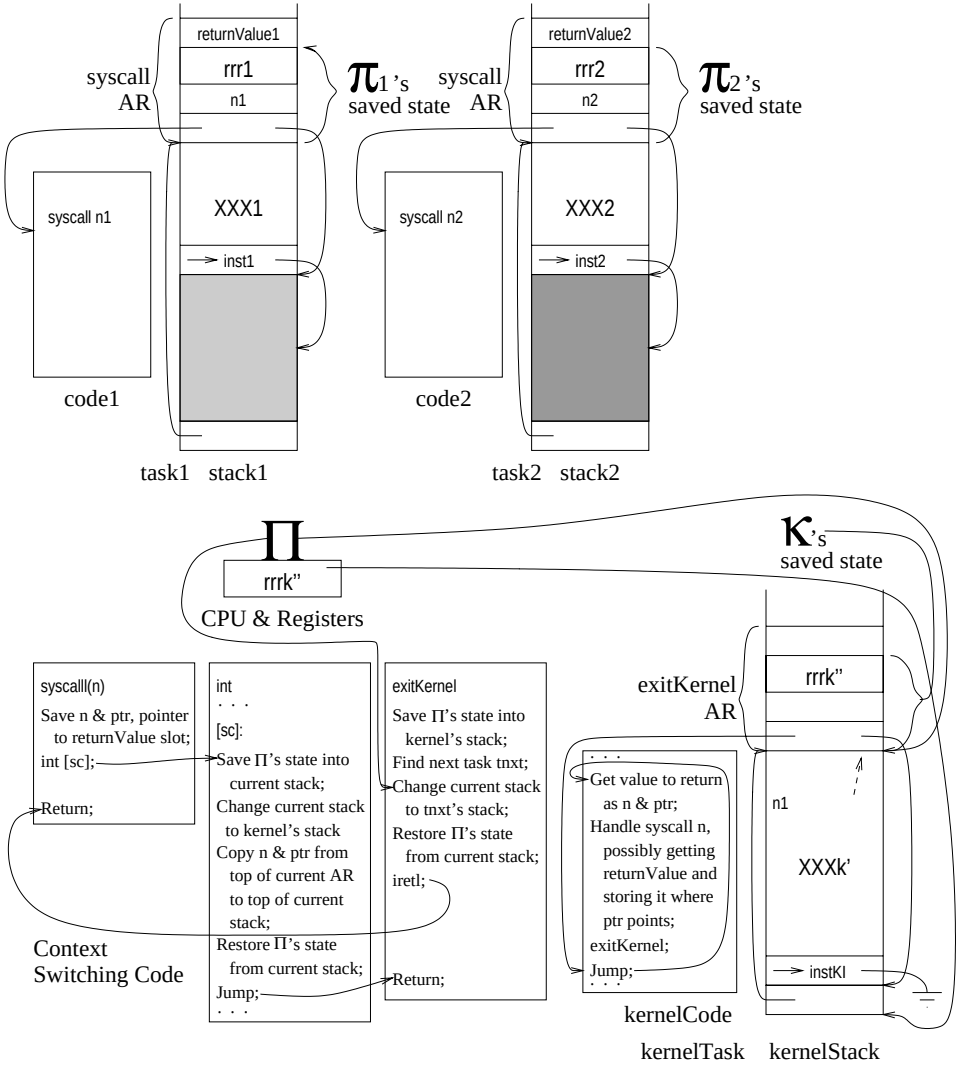

n1
rrr1
n1
syscall
AR
π1 's
saved state
returnValue2
rrr2
n2
syscall
AR
π2 's
saved state
syscall n1
XXX1
inst1
code1
task1 stack1
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrrk
CPU & Registers
κ's
saved state
n1
rrrk
exitKernel
AR
XXXk
instKl
syscalll(n)
Save n & ptr, pointer
to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into
current stack;
Change current stack
to kernel's stack
Copy n & ptr from
top of current AR
to top of current
stack;
Restore Π's state
from current stack;
Jump;
exitKernel
Save Π's state into
kernel's stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
iretl;
Return;
Get value to return
as n & ptr;
Handle syscall n,
possibly getting
returnValue and
storing it where
ptr points;
exitKernel;
Jump;
Context
Switching Code
kernelCode
kernelTask
kernelStack

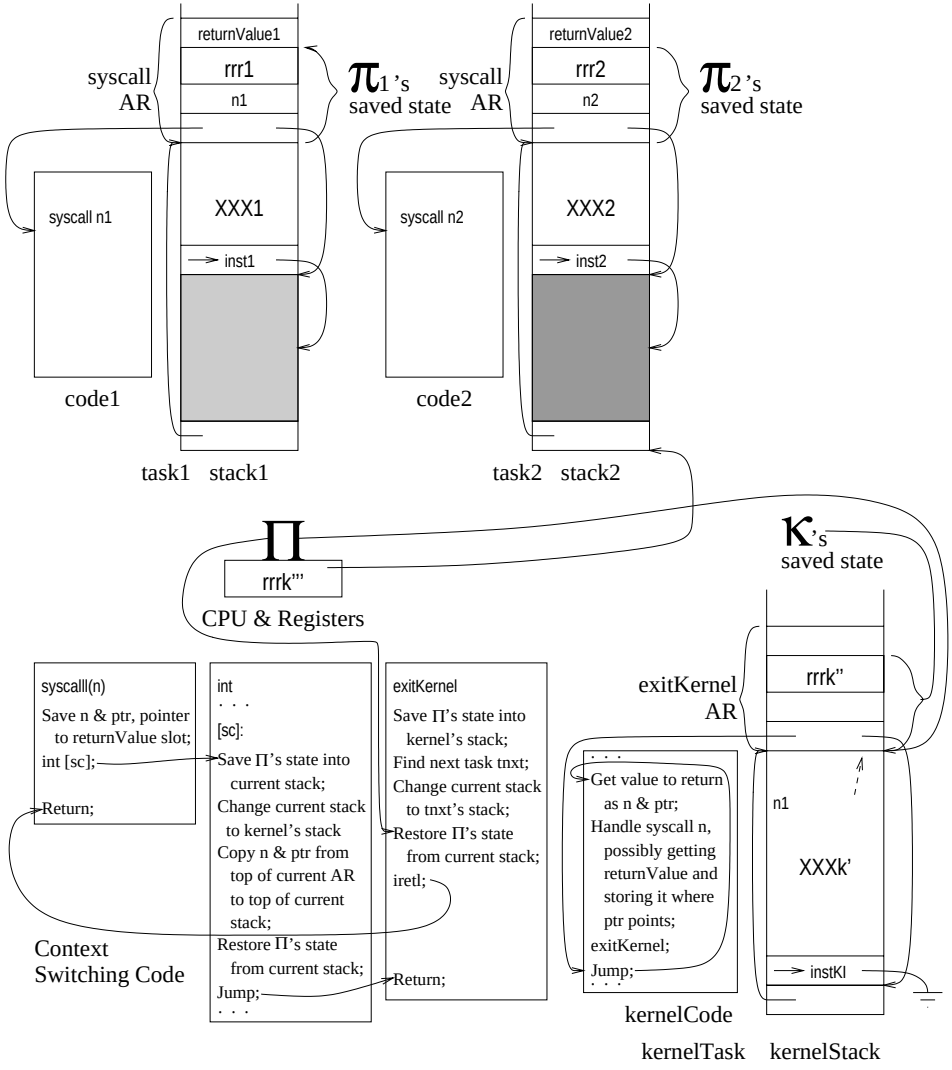

n1
rrr1
n1
syscall
AR
π1 's
saved state
returnValue2
rrr2
n2
syscall
AR
π2 's
saved state
syscall n1
XXX1
inst1
code1
task1 stack1
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrrk
CPU & Registers
syscalll(n)
Save n & ptr, pointer
to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into
current stack;
Change current stack
to kernel's stack
Copy n & ptr from
top of current AR
to top of current
stack;
Restore Π's state
from current stack;
Jump;
exitKernel
Save Π's state into
kernel's stack;
Find next task tnxt;
Change current stack
to tnxt's stack;
Restore Π's state
from current stack;
iretl;
Return;
Get value to return
as n & ptr;
Handle syscall n,
possibly getting
returnValue and
storing it where
ptr points;
exitKernel;
Jump;
n1
XXXk
instKl
Context
Switching Code
kernelCode
kernelTask
kernelStack

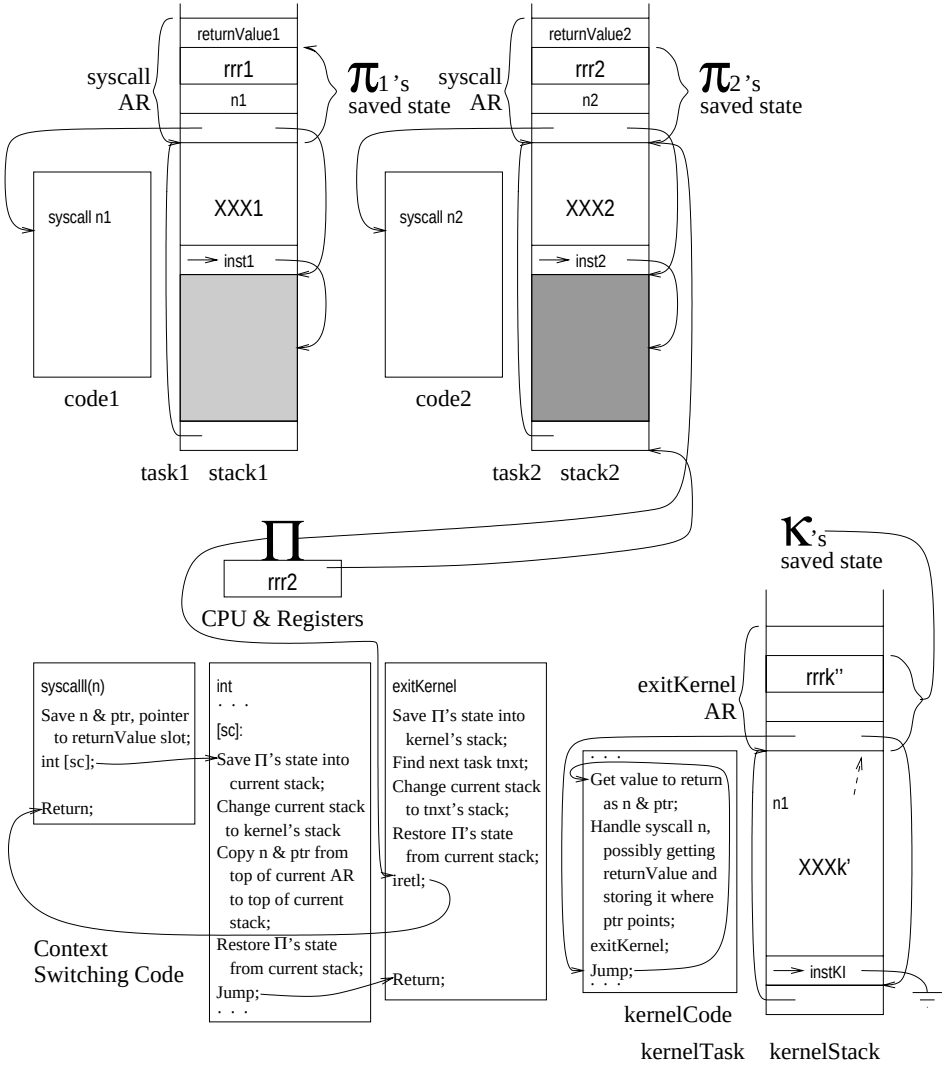

returnValue1
rrr1
n1
syscall AR
π1's saved state
returnValue2
rrr2
n2
syscall AR
π2's saved state
syscall n1
XXX1
inst1
code1
task1 stack1
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrrk'
CPU & Registers
syscalll(n)
Save n & ptr, pointer to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into current stack;
Change current stack to kernel's stack
Copy n & ptr from top of current AR to top of current stack;
Restore Π's state from current stack;
Jump;
exitKernel
Save Π's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore Π's state from current stack;
iretl;
Return;
Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;
n1
XXXk'
instKl
Context Switching Code
kernelCode
kernelTask
kernelStack

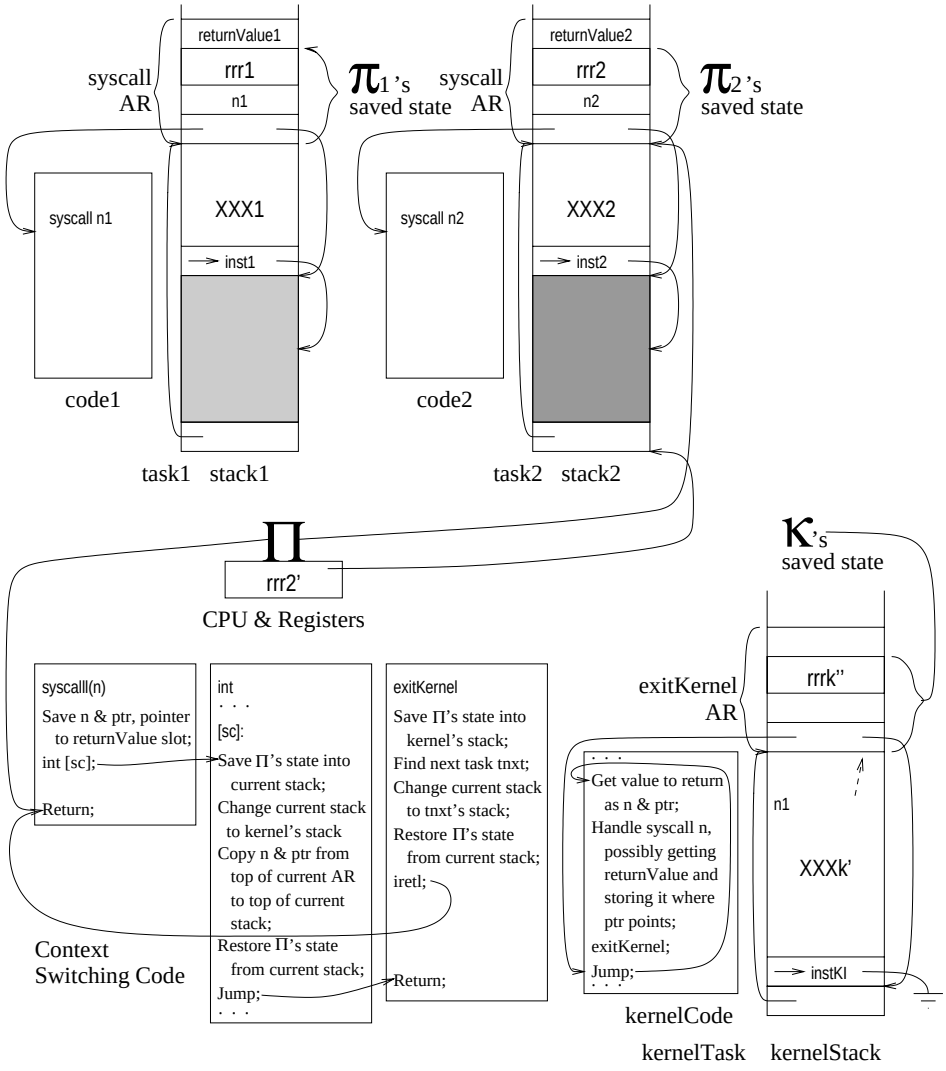

returnValue1
rrr1
n1
syscall AR
π1 's saved state
returnValue2
rrr2
n2
syscall AR
π2 's saved state
syscall n1
XXX1
inst1
code1
syscall n2
XXX2
inst2
code2
task1 stack1
task2 stack2

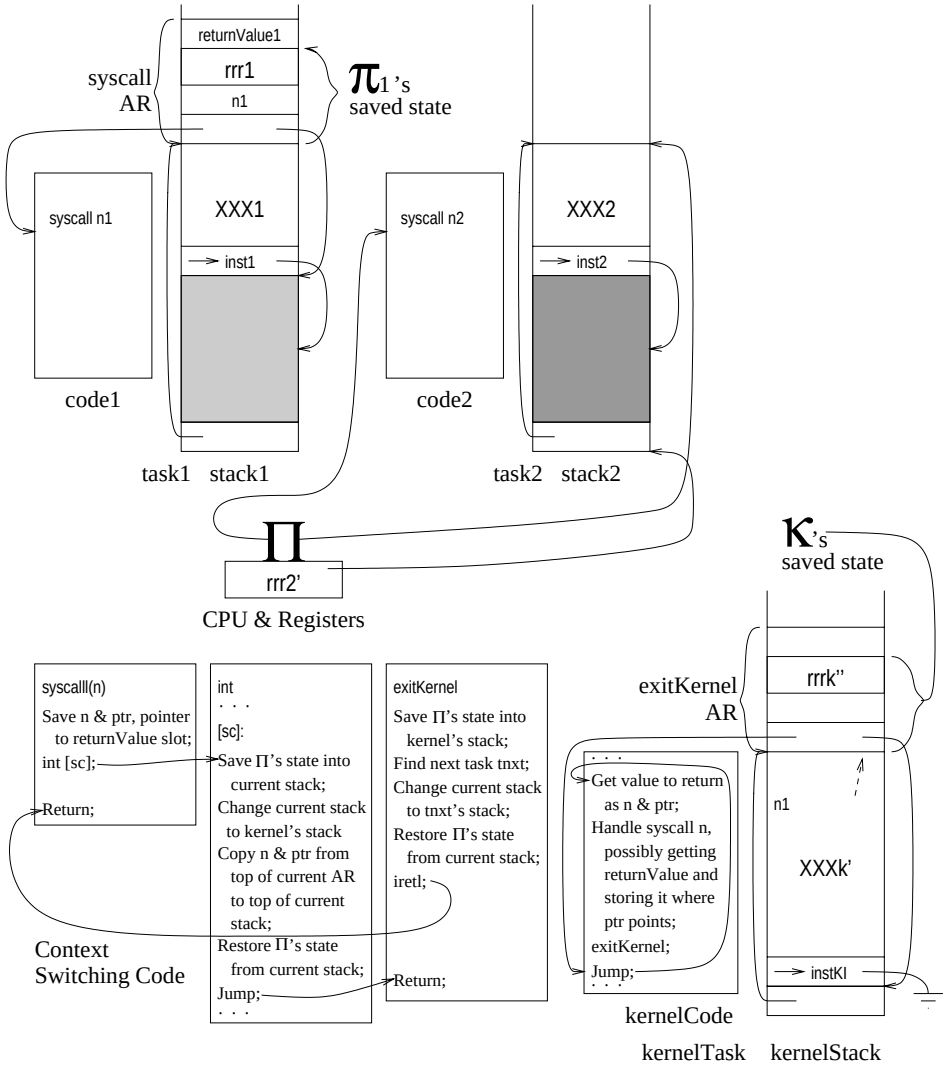

Π
rrrk"
CPU & Registers
syscalll(n)
Save n & ptr, pointer to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into current stack;
Change current stack to kernel's stack
Copy n & ptr from top of current AR to top of current stack;
Restore Π's state from current stack;
Jump;
exitKernel
Save Π's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore Π's state from current stack;
iretl;
Return;
Context Switching Code
exitKernel AR
Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;
kernelCode
n1
XXXk'
instKl
kernelTask
kernelStack


returnValue1
rrr1
n1
syscall AR
π1 's saved state
returnValue2
rrr2
n2
syscall AR
π2 's saved state
syscall n1
XXX1
inst1
code1
task1 stack1
syscall n2
XXX2
inst2
code2
task2 stack2
Π
rrrk"
CPU & Registers
κ's saved state
syscalll(n)
Save n & ptr, pointer to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into current stack;
Change current stack to kernel's stack
Copy n & ptr from top of current AR to top of current stack;
Restore Π's state from current stack;
Jump;
exitKernel
Save Π's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore Π's state from current stack;
iretl;
Return;
Context Switching Code
exitKernel AR
rrrk"
n1
XXXk'
instKl
Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;
kernelCode
kernelTask
kernelStack


returnValue1
rrr1
n1
syscall AR
π1's saved state
returnValue2
rrr2
n2
syscall AR
π2's saved state
syscall n1
XXX1
inst1
code1
syscall n2
XXX2
inst2
code2
task1 stack1
task2 stack2
Π
rrrk'''
CPU & Registers
K's saved state
syscalll(n)
Save n & ptr, pointer to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into current stack;
Change current stack to kernel's stack
Copy n & ptr from top of current AR to top of current stack;
Restore Π's state from current stack;
Jump;
exitKernel
Save Π's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore Π's state from current stack;
iretl;
Return;
Context Switching Code
exitKernel AR
rrrk"
Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;
n1
XXXk'
instKl
kernelCode
kernelTask
kernelStack


returnValue1
rrr1
n1
syscall AR
π1's saved state
returnValue2
rrr2
n2
syscall AR
π2's saved state
syscall n1
XXX1
inst1
code1
syscall n2
XXX2
inst2
code2
task1 stack1
task2 stack2
Π
rrr2
CPU & Registers
κ's saved state
rrrk"
exitKernel AR
n1
XXXk'
instKI
syscalll(n)
Save n & ptr, pointer to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into current stack;
Change current stack to kernel's stack
Copy n & ptr from top of current AR to top of current stack;
Restore Π's state from current stack;
Jump;
exitKernel
Save Π's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore Π's state from current stack;
iretI;
Return;
Context Switching Code
Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;
kernelCode
kernelTask
kernelStack


returnValue1
rrr1
n1
syscall AR
π1's saved state
returnValue2
rrr2
n2
syscall AR
π2's saved state
syscall n1
XXX1
inst1
code1
syscall n2
XXX2
inst2
code2
task1 stack1
task2 stack2
Π
rrr2'
CPU & Registers
κ's saved state
syscalll(n)
Save n & ptr, pointer to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into current stack;
Change current stack to kernel's stack
Copy n & ptr from top of current AR to top of current stack;
Restore Π's state from current stack;
Jump;
exitKernel
Save Π's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore Π's state from current stack;
iretl;
Return;
Context Switching Code
exitKernel AR
rrrk''
n1
XXXk'
instKI
Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;
kernelCode
kernelTask
kernelStack


returnValue1
rrr1
n1
syscall AR
π1's saved state
syscall n1
XXX1
inst1
code1
syscall n2
XXX2
inst2
code2
task1 stack1
task2 stack2
Π
rrr2'
CPU & Registers
κ's saved state
exitKernel AR
rrrk''
n1
XXXk'
instKI
kernelStack
syscalll(n)
Save n & ptr, pointer to returnValue slot;
int [sc];
Return;
int
[sc]:
Save Π's state into current stack;
Change current stack to kernel's stack
Copy n & ptr from top of current AR to top of current stack;
Restore Π's state from current stack;
Jump;
exitKernel
Save Π's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore Π's state from current stack;
iretl;
Return;
Context Switching Code
Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;
kernelCode
kernelTask