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.
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.
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.
Context Switching Code

1. syscall(n)
2. Save n's state into current stack;
3. Handle syscall n, possibly leaving returnValue in current stack;
4. Find next task tnxt;
5. Change current stack to tnxt's stack;
6. Restore n's state from current stack;
7. Return with returnValue;

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>170</td>
<td>130</td>
<td>95</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU & Registers
Handle syscall \( n \), possibly leaving return value in current stack; Find next task \( \text{tnxt} \); Change current stack to \( \text{tnxt}'s \) stack; Restore \( \Pi \)'s state from current stack; Return with return value.

\( \Pi \)'s saved state
Context Switching Code

### CPU & Registers

- syscall AR
- syscall n
- syscall n1
- syscall n2
- syscall n

### Return Value

- returnValue1
- returnValue2

### Saved State

-saved state
- π₁’s saved state
- π₂’s saved state

### Task

- task1
- task2

### Stack

- stack1
- stack2

- syscall(n)
  - Save IT'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 IT’s state from current stack;
  - Return with returnValue;

© 2007 Daniel M. Berry

Real-Time Programming: Trains Pg. 13
“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.
Context Switching Code

CPU & Registers

 syscall n1

 syscall AR

 syscall(r)

 Save IT's state into current stack;
 Handle syscall r, possibly leaving returnValue in current stack;
 Find next task tnext;
 Change current stack to tnext's stack;
 Restore IT's state from current stack;
 Return with returnValue;

 syscall n2

 syscall AR

 syscall(r)

 Save IT's state into current stack;
 Handle syscall r, possibly leaving returnValue in current stack;
 Find next task tnext;
 Change current stack to tnext's stack;
 Restore IT's state from current stack;
 Return with returnValue;

 task1 stack1

 task2 stack2

 task1 code1

 returnValue1

 rrr1

 n1

 syscall n1

 XXX1

 inst1

 task2 code2

 returnValue2

 rrr2

 n2

 syscall n2

 XXX2

 inst2

 Context Switching Code
Context Switching Code

SYSCALL

CPU & Registers

# Context Switching Code

SYSCALL

Save IT'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 IT's state from current stack;
Return with returnValue;

SYSCALL(n)

Return withReturnValue;

Restore IT's state into current stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore IT's state from current stack;
Return with returnValue;

SYSCALL(n)

Return withReturnValue;

Restore IT's state into current stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore IT's state from current stack;
Return with returnValue;

SYSCALL(n)

Return withReturnValue;

Restore IT's state into current stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore IT's state from current stack;
Return with returnValue;

SYSCALL(n)

Return withReturnValue;

Restore IT's state into current stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore IT's state from current stack;
Return with returnValue;

SYSCALL(n)

Return withReturnValue;

Restore IT's state into current stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore IT's state from current stack;
Return with returnValue;

SYSCALL(n)

Return withReturnValue;
Context Switching Code

 syscall(n)  
 Save IT’s state into current stack;  
 Handle syscall n, possibly leaving returnValue in current stack;  
 Find next task nxt;  
 Change current stack to nxt’s stack;  
 Restore IT’s state from current stack;  
 Return with returnValue;

 syscall(n)  
 Current stack;

 syscall(\pi)’s saved state  
 \pi

 syscall(n)

 stack1

 task1

 code1

 syscall(n2)

 stack2

 task2

 code2

 syscall(\pi)

 returnValue

 CPU & Registers

  2007 Daniel M. Berry

 Real-Time Programming: Trains  Pg. 20
Low Level View

Unbounded number of processes, one CPU, and kernel task

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

Context Switching Code

- syscall(n)
  - Save n & ptr, pointer to return Value slot;
  - int [sc];
  - Return;

- exitKernel
  - Save IT’s state into kernel’s stack;
  - Find next task tnxt;
  - Change current stack to tnxt’s stack;
  - Restore IT’s state from current stack;
  - iretl;
  - Return;

- Get value to return as n & ptr;
  - Handle syscall n, possibly getting return Value and storing it where ptr points;
  - exitKernel;
  - Jump;

- kernelCode

- .
- ....

- .
- ...

- Copy n & ptr from top of current AR to top of current stack;

- Context Switching Code
Get value to return as n & ptr; Handle syscall n, possibly getting returnValue and storing it where ptr points; exitKernel.

Running

syscall AR

RRR1

Saved state

syscall n1

code1

task1

stack1

Context Switching Code

syscall(n)

Save n & ptr; pointer to returnValue slot; int [sc];

Return.

kernelCode

kernelTask

kernelStack

exitKernel

Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel;

Jump;

ιπ2's saved state

syscall n2

code2

task2

stack2

CPU & Registers

int inst1

XXX1

Copy n & ptr from top of current AR to top of current stack; Restore IT's state from current stack; iretl;

Jump;

Return;

ιπ2

rrr2

rrrK

exitKernel

Save IT's state into kernel's stack;
Find next task nxt; Change current stack to nxt's stack; Restore IT's state from current stack; iretl;

Jump;

Return;

κ's saved state

syscall n1

code1

task1

stack1

Context Switching Code

syscall(n)

Save n & ptr; pointer to returnValue slot; int [sc];

Return.
Get value to return as n & ptr; Handle syscall n, possibly getting returnvalue and storing it where ptr points; exitKernel; . . .

Save n & ptr, pointer to returnvalue slot; int [sc]; Return;

syscall(n)
Save n & ptr, pointer to returnvalue slot; int [sc];
Return;

Context Switching Code

ExitKernel
Save PI's state into kernel's stack; Find next task txt; Change current stack to txt's stack; Restore PI's state from current stack; iretl;

Get value to return as n & ptr; Handle syscall n, possibly getting returnvalue and storing it where ptr points; exitKernel; Jump;

Kernel's state

Syscall Code

Syscall
Save syscall'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 PI's state from current stack; iretl;

Context Switching Code

Syscall
Save syscall'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 PI's state from current stack; iretl;

Context Switching Code

Syscall
Save syscall'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 PI's state from current stack; iretl;

Context Switching Code

Syscall
Save syscall'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 PI's state from current stack; iretl;

Context Switching Code

Syscall
Save syscall'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 PI's state from current stack; iretl;

Context Switching Code

Syscall
Save syscall'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 PI's state from current stack; iretl;

Context Switching Code

Syscall
Save syscall'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 PI's state from current stack; iretl;
Restore 's state

kernelCode

context

switching code

real-time programming: trains pg. 27
Context Switching Code

syscall(n)
Save n & ptr, pointer to returnValue slot
int [sc];
Return;

Get value to return
as n & ptr;
Handle syscall n,
possibly getting
returnValue and
storing it where
ptr points;
exitKernel;

kernelCode
kernelTask
kernelStack

CPU & Registers

exitKernel
Save IT's state into
kernel's stack;
Find next task tntx;
Change current stack
to tntx's stack;
Restore IT's state
from current stack;
iretl;

Save IT's state into
current stack;
Copy n & ptr from
top of current AR
to top of current
stack;
Restore IT's state
from current stack;
Jump;

syscall(n)
Save n's state into
kernel's stack;
Change current stack
to kernel's stack;

Get value to return
as n & ptr;
Handle syscall n,
possibly getting
returnValue and
storing it where
ptr points;
exitKernel;

Save IT's state into
current stack;
Change current stack
to AR's stack;
Restore IT's state
from current stack;
iretl;

Save n's state into
AR's stack;
Change current stack
to AR's stack;
Restore IT's state
from current stack;
iretl;
kernelTask

Jump;

kernelCode

Get value to return as n & ptr; Handle syscall n, possibly getting returnValue and storing it where ptr points; exitKernel;

.. . .

Context Switching Code

syscall(n)
Save n & ptr, pointer to returnValue slot; int [sc]; Return;

CPU & Registers

getTask

Save n'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 n's state from current stack; iret;

Return;

Get value to return as n & ptr; Handle syscall n, possibly getting returnValue and storing it where ptr points; exitKernel; Jump;

kernelCode

kernelTask

kernelStack
Context Switching Code

syscall \( n \)
- Save \( n \) & ptr, pointer to returnValue slot
- int [sc]:
  - Return;

CPU & Registers

Get value to return as \( n \) & ptr;
Handle syscall \( n \), possibly getting returnValue and storing it where ptr points;
exitKernel;
Jump;

kernelCode
kernelTask
kernelStack

Real-Time Programming: Trains  Pg. 31 © 2007 Daniel M. Berry
Context Switching Code

syscall(n)
- Save n & ptr, pointer to returnValue slot;
- int [sc];
- Return;

CPU & Registers

syscall(n)
- Save n & ptr, pointer to returnValue slot;
- int [sc];
- Return;

exitKernel
- Save IT'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 IT's state from current stack;
- ireti;
- Return;

exitKernel
- Get value to return as n & ptr;
- Handle syscall n, possibly getting returnValue and storing it where ptr points;
- exitKernel;
- Jump;

kernelTask
- kernelCode
- kernelStack

n1
- syscall(n1)
- XXX1
- inst1
- code1
- task1
- stack1

n2
- syscall(n2)
- XXX2
- inst2
- code2
- task2
- stack2

rrr1
- syscall AR
- Saved state

rrr2
- syscall AR
- Saved state

rrr1
- syscall AR
- Saved state

rrr2
- syscall AR
- Saved state

MRK
- syscall AR
- Saved state

© 2007 Daniel M. Berry

Real-Time Programming: Trains Pg. 32
syscall(n) int exitKernel

Context Switching Code

instKI

kernelCode

kernelTask

kerneStack

Get value to return as n & ptr; Handle syscall n, possibly getting returnValue and storing it where ptr points; exitKernel;
Context Switching Code

syscall n

Save n & ptr, pointer to returnValue slot; int [sc]:...

Return;

CPU & Registers

syscall n

Save n & ptr, pointer to returnValue slot; int [sc]:...

Return;

exitKernel

Save Π's state into kernel's stack; Find next task tnx; Change current stack to tnx’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:...

kernelCode

kernelTask

kernelStack
Context
Switching Code

cpu & registers

syscall(n)

Save n & ptr, pointer to returnValue slot; int [sc];
Return;

exitKernel

Save IT's state into kernel's stack;
Find next task tnxt;
Change current stack to tnxt's stack;
Restore IT's state from current stack;
iretl;
Return;

kernelCode

kernelTask

kernelStack
kernelStack
Jump;
kernelCode

Get value to return as n & ptr;
Handle syscall n, possibly getting returnValue and storing it where ptr points;
exitKernel

..

n1
syscall(n)
int exitKernel

...[sc]:
kernelTask
task2
syscall n2
n2
returnValue2
's saved state
rrr2
π
saved state
syscall
AR
stack1
stack2

Context Switching Code

syscall(n)
Save n & ptr, pointer to returnValue slot; int [sc];
Return;

CPU & Registers

int
[sc];
Save II'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 II's state from current stack;
iretl;

exitKernel
Save II's state into kernel's stack;
Find next task tnx;
Change current stack to tnx's stack;
Restore II's state from current stack;
iretl;

Return;

kernelCode
kernelTask
kernelStack

© 2007 Daniel M. Berry

Real-Time Programming: Trains Pg. 36
Handle syscall $n_1$, possibly getting $returnValue$ and storing it where $ptr$ points; exitKernel.

\[ \text{syscall}(n) \quad \text{int} \quad \text{exitKernel} \]

Save 's state into current stack; Change current stack to kernel's stack.

\[ \text{task1 stack1} \quad \text{code1} \]

\[ \text{tau} \quad \text{lambda} \quad \text{instKI} \]

\[ \text{kernelTask} \quad \text{kernelStack} \]

Jump;

Get value to return kernel's stack; Jump;

\[ \text{Context Switching Code} \]

\[ \text{intValue1} \quad \text{pi1's saved state} \]

\[ \text{syscall n2} \quad \text{task2 stack2} \]

\[ \text{rcode2} \quad \text{task2} \]

\[ \text{iretl} \]

\[ \text{fullName2} \quad \text{fullName1} \quad \text{fullName0} \]

\[ \text{CPU & Registers} \]

Find next task $\text{tnxt}$; Change current stack to $\text{tnxt}$'s stack; Restore 's state from current stack; iretl;

\[ \text{kernelCode} \quad \text{exitKernel} \quad \text{AR} \]

\[ \text{rrrk} \]

\[ \text{saved state} \]

\[ \text{rrr2} \]

\[ \text{stack; \quad stack; \quad \text{returnValue1}} \]

\[ \text{returnValue1} \]

\[ \text{rrr1} \quad \text{pi1's saved state} \]

Save $n_1$ & $ptr$, pointer to $returnValue$ slot; int [sc]; Return;

\[ \text{Context Switching Code} \]

\[ \text{syscall(n)} \quad \text{int} \quad \text{exitKernel} \]

Save II'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 II's state from current stack; iretl; Return;

\[ \text{kernelCode} \quad \text{exitKernel} \quad \text{AR} \]

\[ \text{rrmk} \]

\[ \text{saved state} \]

\[ \text{rrr1} \]

\[ \text{XXX1} \quad \text{XXX2} \]

\[ \text{fullName1} \quad \text{fullName2} \]

\[ \text{fullName0} \]