Taskserver Design

Jan Tepelmann Marcel Noe Moritz Lapp

SDI Group $\log_2(256)$
Universität Karlsruhe (TH)

System Design & Implementation, 2008

Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

System Components

L4 Microkernel

Sigma 0

Anonymous Memory Provider

Syscall Server

Data Space Providers

ELF Loader

Fileserver

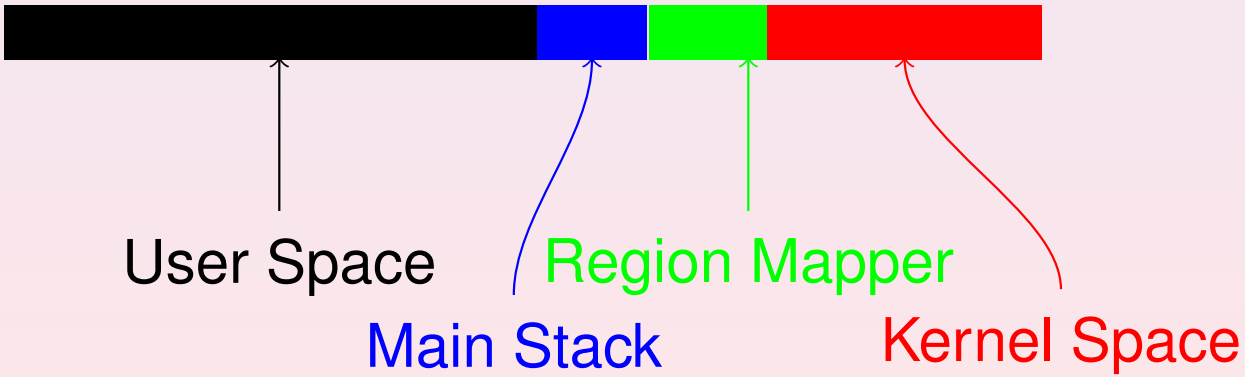
Taskserver

Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

Sawmill Inspired Data Spaces

- Every address space has got it's own managing thread, called *region mapper*
- *region mapper* resides at the end of user address space, just below kernel
- *region mapper* holds mapping between *VM Region* and *Data Space Provider*

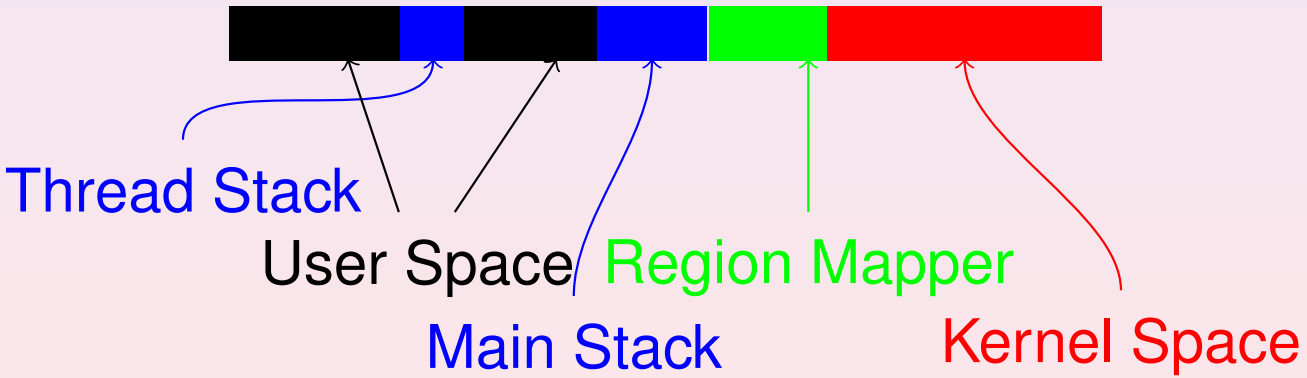


Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

Stack Positioning

- *Main program stack* is created just below *Region Mapper*, growing down, towards *heap*
- For every additional thread, *stack space* is allocated from *heap*, surrounded by *read only pages* to detect overflow
- *Thread stacks* are created by *region mapper*



Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

Process management

- *startTask()*
- *fork()*
- *exec()*
- *kill()*
- *waitTid()*
- *createThread()*
- *getThreadStatus()*

startTask

```
L4_ThreadId_t startTask(in String path, in
String args);
```

- *Task server* creates a new *region mapper* inside its new *address space* using the *syscall server*
- *Task server* configures the *address space*
- *Task server* sends message to *Region mapper*, telling it the path of the image to load
- *Region mapper* asks *ELF-Loader* (or *PE-Loader* or whatever) to map image into its *address space*
- *Region mapper* starts mapped program inside new thread

fork

```
L4_ThreadId_t fork();
```

- *Task server* asks *memory server* to create a new *address space*
- *Task server* creates a new *region mapper* inside new *address space*
- *Task server* asks *region mappers* to map old *User space* and *Stack* into new *address space*
- *Task server* sends message to *region mapper*
- *Region mapper* resumes operation in new address space

exec

```
L4_ThreadId_t exec(in String path, in String args);
```

- *Task server* kills alll *threads* inside address space except *region mapper*
- *Task server* sends message to *Region mapper*, telling it the path of the image to load
- *Region mapper* asks *ELF-Loader* (or *PE-Loader* or whatever) to map image into its *address space*
- *Region mapper* starts mapped program inside new thread

kill

```
Void kill(in L4_ThreadId_t tid);
```

- If *TID* is a region mapper: Kill all *threads* in *address space*
- Else kill *thread* specified by *TID*
- *Suicidal tendencies*: Every thread has to kill itself at the end

waitTid

```
L4_Word_t waitTid(in L4_ThreadId_t tid);
```

- Waits for *thread* specified by *tid* to terminate
- Returns the *thread's exit* status
- Status information is stored in *region mapper* of *thread* until *waitTid()* is called

createThread

```
L4_ThreadId_t createThread();
```

- *createThead()* returns *global thread id* of new *thread*.
- *Task server* tells *syscall server* to create a new *thread* inside specified *address space*
- *Task server* tells *region mapper* to start thread
- *Region mapper* creates *thead stack* and sends start message to *thread*

getThreadStatus

```
L4_Word_t getThreadStatus(in L4_ThreadId_t tid);
```

- Uses *schedule()* system call to check whether a *thread* is still alive. Returns the status code which is returned by *schedule*.

Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

Settings, Statistics & Status

- *setStatisticInterval()*
- *setTimeslice()*
- *setPriority()*
- *setPreemptionDelay()*

setStatisticInterval

```
void setStatisticInterval(in L4_Word_t
interval);
```

- *interval* is specified in milliseconds
- Sets interval in which statistics are collected
- Uses *schedule* syscall and an ugly hack which uses the total quantum

setTimeslice

```
void setTimeslice(in L4_Word_t timeslice);
```

- *timeslice* is specified in milliseconds
- Sets length of timeslice

setPriority

```
L4_Word_t setPriority(in L4_Word_t priority);
```

- Sets priority of *thread* identified by *TID*

setPreemptionDelay

```
L4_Word_t setPreemptionDelay(in L4_ThreadId_t
tid, in L4_Word_t sensitivePrio, in L4_Word_t
maxDelay);
```

- Sets preemption delay of *thread* identified by *TID*

Outline

1. Design
   - System Components
   - Sawmill Inspired Data Spaces
   - Stack Positioning
2. Interface
   - Process management
   - Settings, Statistics & Status
3. Statistics
   - Statistics over virtual Filesystem

Statistics

- Collected statistics are accessed via virtual filesystem

Questions?

- Please feel free to ask *questions* or give *comments*!