TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Exercise 2: IPC

Inter-Process Communication

Recap

- Inter-Process Communication
  - Send
  - Wait (open/closed)
  - Call: Send + closed Wait
  - Replay and Wait
- Message Payload
  - Plain data : copy
  - Capabilities (memory, kernel objects) : map
- Sync / Async
  - Sync: Rendezvous, direct copy sender → receiver
  - Async: Queues, buffers, fire and forget

The Plan

- IPC – call & reply
  - Create a second thread
  - Send an empty message, wait for answer
  - Send numbers, get the sum as result
  - Math server: opcode & operands
- Log server
  - Use UTCB as string buffer
  - Create 10 threads, all printing their ID

Setup

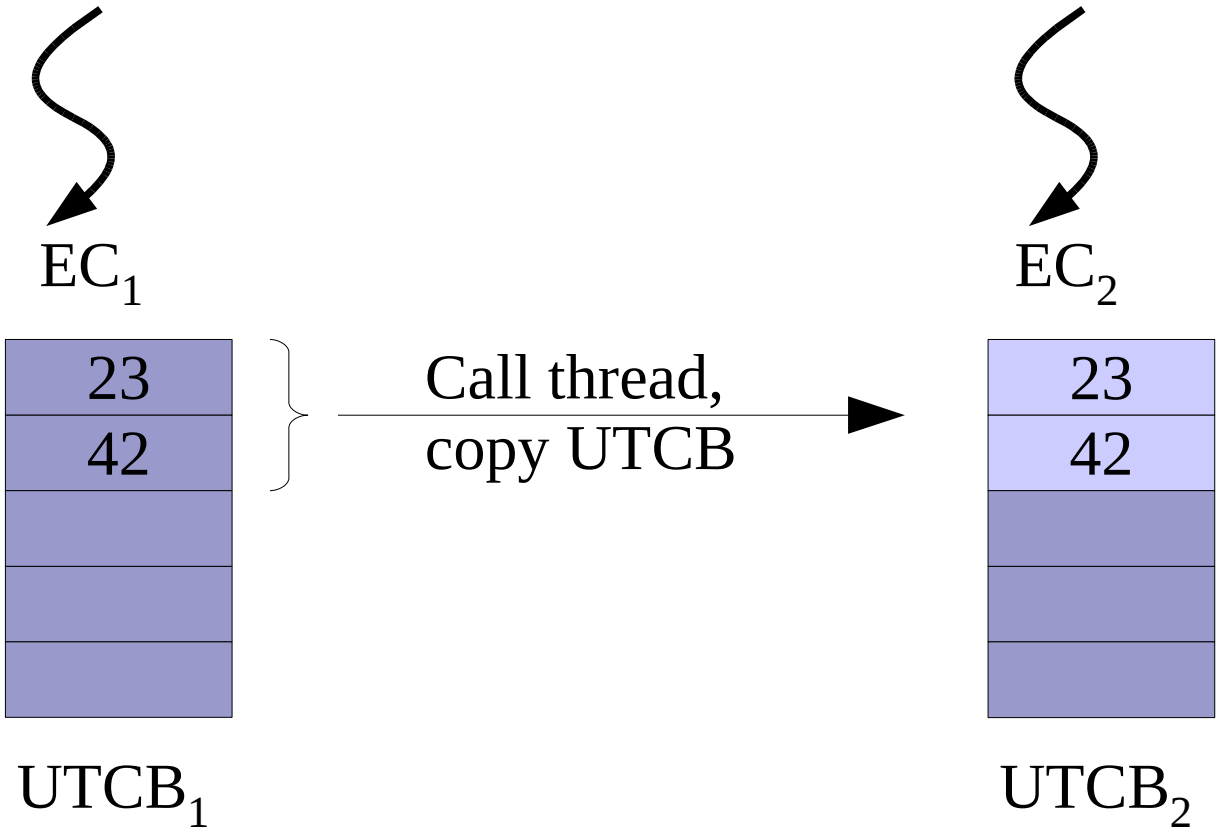
- Download the source archive from
  `http://os.inf.tu-dresden.de/Studium/KMB/WS2010/Exercise2.tar.bz2`
- Unpack, compile, run
  ```
  cd iso; make; make run
  ```

- user/src/ipc.cc : primary file to edit
- user/include/nova.h : system call bindings
- kern/doc/specification.pdf : interface spec

NOVA Basics

- Execution Context (EC)
  - Some unit of execution
  - Thread or virtual machine
- Scheduling Context (SC)
  - Unit of Time with a priority
  - Determines which thread will run
- Portal (PT)
  - Communication endpoint
  - Messages are sent to portals, threads wait there to receive

IPC Basics



Exercise: ipc.cc – Initialization

- Some CPU counting, UTCB, capability space
- Initialize serial port for logging output
- Scan memory, find suitable block (Stack)
- Program complete, enter when ready ...

```
Unsigned caps : start of free cap slots
Stack* stack : Memory area usable as stacks
Utcb* utcb : Message transfer via UTCB
```

Thread Creation

```
// allocate 2 cap slots
unsigned cap_ec = caps++;
unsigned cap_pt = caps++;

// create new execution context
create_ec (cap_ec, utcb--, stack++);

// create portal with thread in it
create_pt (cap_pt, cap_ec, thread_echo);

// call thread through portal
call (cap_pt, Mtd());
```

Thread Calling

```
// code to run when calling the thread
REGPARAM(1)
static unsigned thread_echo(uint32, Utcb*)
{
   printf (“hi there\n”);
   return 0;
}

// call thread through the portal
call (cap_pt, Mtd());
```

IPC – Message Transfer


```
utcb->msg[0] = 23
utcb->msg[1] = 42
call (cap_pt, Mtd(2,0));
```

Message Transfer Descriptor

- Number of raw data words: 2
- Number of map items: 0 (for now)

Various Servers

- Write your sum thread
  - msg[0] and msg[1] : operands
  - Call portal, thread calculates sum in msg[2]
  - Reply with all three words to the sender
- Write your math thread
  - Add opcode (ADD, SUB, MUL, DIV)
  - Thread performs operation and replies
- Log server
  - Use msg area to transfer a const string
  - Server printf it

Log Client Setup

```
// allocate 2 cap slots
unsigned cap_ec = caps++;
unsigned cap_sc = caps++;

// create new execution context
create_ec2 (cap_ec, utcb, stack++,
  thread_client, 0, 0, false);
utcb->msg[0] = cap_pt;
utcb->msg[1] = <some id>;

// create SC with 1ms and priority 1
create_sc (cap_sc, cap_ec, Qpd(1000,1));
utcb--;
```

Log Client Thread

```
REGPARAM(1) static unsigned
thread_client(uint32, Utcb* utcb)
{
   unsigned cap_pt = utcb->msg[0];
   unsigned id = utcb->msg[1];

   char const * text = "i am thread _";
   unsigned len = strlen(text);

   memcpy (utcb->msg, text, len + 1);
   len = (len + 3) / 4;
   call (cap_pt, Mtd (len, 0));
}
```