Security Engineering
6th Problem Set

Prof Stefan Lucks, Nathalie Jolanthe Dittrich
<first>.<middle>.<lastname>(at)uni-weimar.de

Bauhaus-Universität Weimar

Jan 18, 2019

Section 1

Sorting in `gnatprove`

Task 2

Mini Project

Section 2

Tasks in Ada - Parallel Sum

Tasks

Recap from the Lecture

- Definition as `task` or `task type`
- Entries:
  - Task objects run immediately after generation
  - There is always an environment task

```
task Worker_Type;
task type Worker_Type is
    -- ..
end Worker_Type;
```

Rendezvous: Accepting Entries

- During a rendezvous, an `accept` statement is executed
- For task synchronization and passing parameters
- Blocking:
  - Restrict code in `accept` .. `end` to copying parameters
  - No intensive actions

```
task type Worker_Type is
    entry Start(Input: Array_Access_Type; F: Natural; T: Natural);
end Worker_Type;

task body Worker_Type is
    A: Array_Access_Type;
    From: Natural;
    To: Natural;
begin
    loop
        accept Start(Input: Array_Access_Type; F: Natural; T: Natural) do
            A := Input;
            From := F;
            To := T;
        end Start;
        -- Do heavy computations
    end loop;
end Worker_Type;
```

Task Management

```
procedure Parallel_Sum(Input: Array_Access_Type;
                       Result: out Item_Type) is
    begin
        if Is_Empty(Input) then
            Result := Integer_To_Item_Type(0);
        elsif Input.all'Length < NUM_WORKERS then
            Result := Do_Sequential_Sum(Input);
        else
            Result := Do_Parallel_Sum(Input);
        end if;
    end Parallel_Sum;
```

Task Management

```
function Do_Parallel_Sum(Input: Array_Access_Type) return Item_Type is
begin
    Logger.Reset;
    Start_Workers(Input);

    while not Logger.Has_Finished loop
        delay DEFAULT_DELAY;
    end loop;

    Destroy_Workers;
    return Logger.Get_Sum;
end Do_Parallel_Sum;
```

Task Management

```
procedure Start_Workers(Input: Array_Access_Type) is
        From: Natural := Input.all'First;
        To:   Natural;
        Part_Length: constant Natural := Input.all'Length / NUM_WORKERS;
    begin
        for I in Workers'range loop
            To := From + Part_Length - 1;

            if I = Workers'Last then
                To := Input.all'Last;
            end if;

            Workers(I) := new Worker_Type;
            Workers(I).Start(Input, From, To);

            From := To + 1;
        end loop;
    end Start_Workers;
```

Task Management

```
accept Start(Input: Array_Access_Type;
                        Input_From: Natural;
                        Input_To: Natural) do
            A := Input;
            From := Input_From;
            To := Input_To;
        end Start;

        if To >= From then
            Sum := A.all(From);
        end if;

        for I in From+1 .. To loop
            Sum := Sum + A.all(I);
        end loop;

        Logger.Add(Sum);
        Logger.Set_Worker_Finished;
```

Protected Types

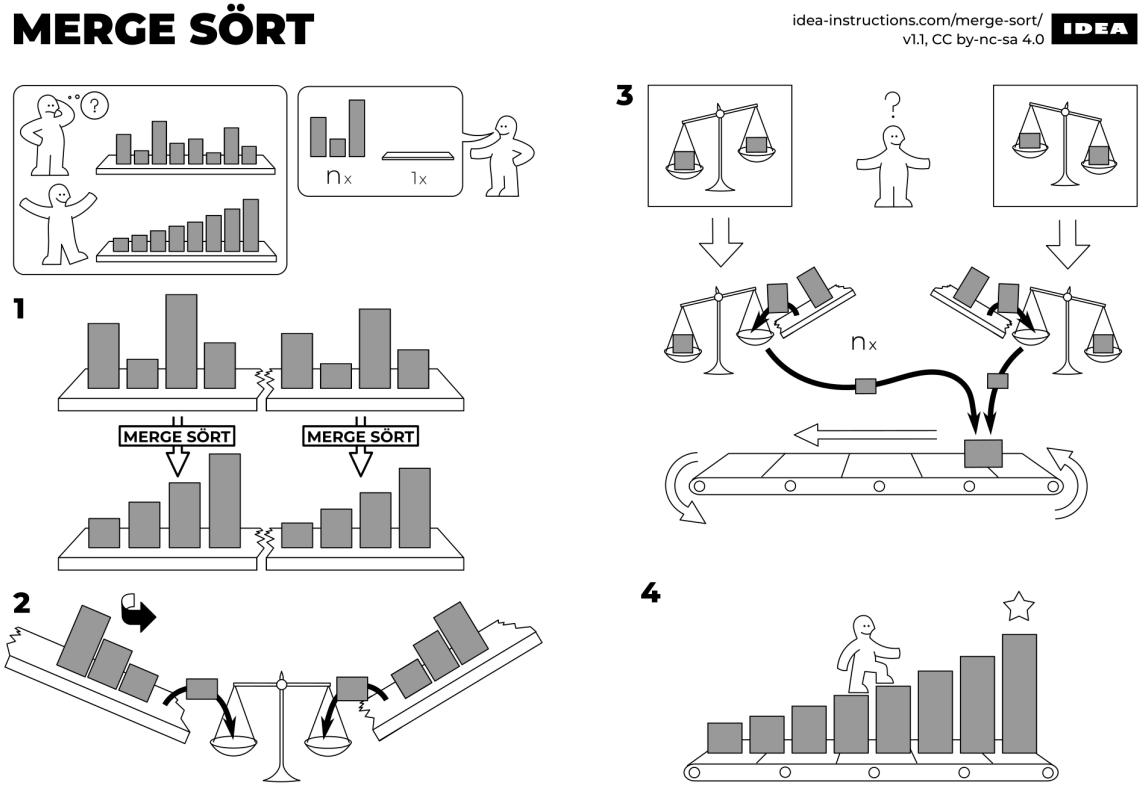
- Encapsulates a shared object
- Can be safely accessed by tasks
- If task executes a `procedure`/an `entry` (may change state), all other operations are blocked:
  - Code therein should be fast
  - **Must avoid** blocking and potentially blocking operations
- `Functions` are non-blocking, **mustnot** change state

```
protected type Logger_Type is
    procedure Add(Partial_Sum: Item_Type);
    function  Get_Sum return Item_Type;
    function  Has_Finished return Boolean;
    procedure Reset;
    procedure Set_Worker_Finished;
private
    Sum: Item_Type;
    Num_Workers_Running: Natural := NUM_WORKERS;
end Logger_Type;

protected body Logger_Type is
    -- ...
end Logger_Type;
```

Section 3

CommandLine Parameters and IO - Concurrent Sorting

Merge Sort


MERGE SÖRT
idea-instructions.com/merge-sort/
v1.1, CC by-nc-sa 4.0
IDEA
nx
1x
1
MERGE SÖRT
MERGE SÖRT
2
3
?
nx
4

CommandLine Parameters

```
with GNAT.Command_Line; use GNAT.Command_Line;
with GNAT.Strings;      use GNAT.Strings;

procedure Main is
    type Context_Type is record
        In_File_Path: aliased String_Access;
        Out_File_Path: aliased String_Access;
        Max_Run_Time: aliased Integer;
    end record;

    Config: Command_Line_Configuration;
    Context: Context_Type;

    procedure Config_Command_Line(Config: in out Command_Line_Configuration;
                                  Context: in out Context_Type) is
    begin
        Define_Switch(
            Config,
            Context.In_File_Path'access,
            "-i:",
            Help => "File path of an input file containing one integer per line.
        );
        -- ...
    end Config_Command_Line;
begin
    Config_Command_Line(Config, Context);
    -- ...
end Main;
```

```
function Read_File(File_Path: String) return Array_Access_Type is
    File: File_Type;
        File_Size: constant Natural := Get_Num_Items_In_File(File_Path);
        A: constant Array_Access_Type := new Array_Type(0 .. File_Size - 1);
    Item: Item_Type;
    begin
    Open(File, In_File, File_Path);

    for I in A'Range loop
      Read(File, Item);
      A.all(I) := Item;
    end loop;

    Close(File);
    return A;
    end Read_File;
```

```
procedure Write_To_File(File_Path: String; A: Array_Type) is
        File: File_Type;
    begin
        if not Exists(File_Path) then
            Create(File, Out_File, File_Path);
        else
            Open(File, Out_File, File_Path);
        end if;

        for I in A'Range loop
            Write(File, A(I));
        end loop;

        Close(File);
    end Write_To_File;
```

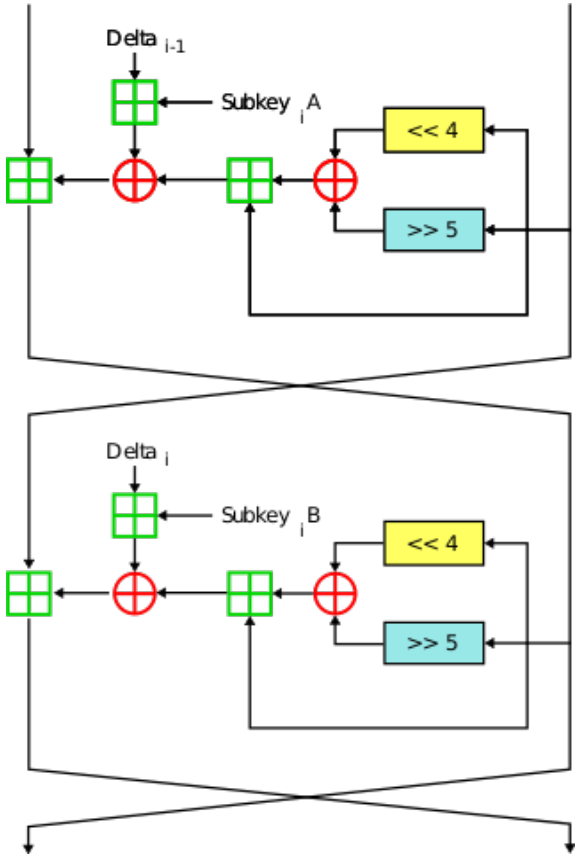
Task Abortion

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Text_IO;             use Ada.Text_IO;

procedure Main is
    ENVIRONMENT_TASK: constant Task_ID := Current_Task;
    MAX_RUN_TIME_IN_SECONDS: constant Duration := 15;
    USER_INPUT_DELAY_IN_SECONDS: constant Duration := 0.1;

    task User_Input_Task;
    task body User_Input_Task is
        Available: Boolean;
        Char: Character;
    begin
        loop
            Get_Immediate(Char, Available);
            if Available and then (Char = 'q' or else Char = 'Q') then
                Abort_Task(ENVIRONMENT_TASK);
            else
                delay USER_INPUT_DELAY_IN_SECONDS;
            end if;
        end loop;
    end User_Input_Task;
begin
    select
        delay MAX_RUN_TIME_IN_SECONDS;
    then abort
        -- Do things
    end select;
    Abort_Task(ENVIRONMENT_TASK);
end Main;
```

Section 4

MITM

XTea



Hints

- For simple programs: simply abort environment task if necessary
- Use the `Armaggedon` package to detect abnormal task terminations
- Print usage information when users provide invalid information
- Keep the code testable

Section 5

Problem Set 7

Recap: What You Already Learnt

- Get to know basics of Ada: ✓
- Get familiar with at least one testing tool: ✓
- Learn how-to do white-box and black-box testing: ✓
- Practicing test coverage: ✓
- Get familiar with formal verification logic: ✓
- Get familiar with access types and data structures: ✓
- Get familiar with `gnatprove`: ✓
- Learn tasks in Ada: ✓

Coming up

- A little more on tasks in Ada
- Some nice algorithms =)
- Plus in the lecture:
  - Logical clocks
  - Resource-sharing algorithms
  - Language-theory and parsing

Final Exams

- Oral Exam
- 04.02.2019-08.02.2019
- 18.03.2019-22.03.2019
- Registration up from next week in the office from Maria-Theresa Hansens (B11 R 113)
- If you are absent in both weeks, please contact us!

Questions?